Belajar Bahasa Pemrograman Python v.1.0

Kholid Fuadi
@sopier
Jogja
2012

Daftar Isi

Pendahuluan

Sebagai seseorang yang tidak memiliki latar belakang formal dalam dunia IT sama sekali, faktanya bahkan saya seorang sarjana ekonomi, menulis buku tentang pemrograman bukanlah perkara mudah, namun demi semangat untuk berbagi, apa yang saya ketahui dan pelajari saya coba untuk bagikan dalam buku ini. Sejak kecil saya memang sudah dekat dengan dunia komputer dan teknologi, kemudian merasa semakin dekat ketika berkenalan dengan dunia *open source*. Saya [masih] terkagum-terkagum dengan semangat para aktivis dunia *open source*, yang rela untuk berbagi, bergotong-royong tanpa harus ada motivasi materi dibelakangnya. Berkenalan dengan dunia *open source* ketika itu tertarik mencoba sistem operasi Linux (tahun 2001, distro Mandrake). Setelah itu aktif mencari tahu dengan semua hal yang berkaitan dengan dunia *open source*, termasuk diantarnya adalah mulai menggunakan aplikasi-aplikasi *open source*, yang berlanjut sampai sekarang, termasuk buku ini pun ditulis menggunakan LaTeX(http://www.latex-project.org), sebuah program *document preparation system* yang sangat ampuh dan *powerful*, dan VIM (http://www.vim.org) sebagai *text editor*-nya.

Berkenalan dengan dunia pemrograman sekitar tahun 2007, ketika belajar membuat aplikasi web (PHP) sederhana. Itu pun tidak sampai terlalu dalam, karena rata-rata aplikasi web yang saya kembangkan menggunakan CMS seperti Joomla, Wordpress, osCommerce, dan lain-lain. Python sendiri merupakan bahasa pemrograman kedua yang saya pelajari setelah PHP. Salah satu fitur yang membuat saya jatuh cinta terhadap Python adalah *built-in interpreter* atau dikenal juga dengan sebutan *shell*, yang berguna untuk melakukan testing awal secara cepat terhadap program yang sedang kita buat. Hal kedua yang membuat saya tertarik dengan Python adalah kenyataan bahwa bahasa ini digunakan secara intensif di perusahaan raksasa sekelas Google, pastinya ada sesuatu mengapa Python dipilih oleh Google.

Pertimbangan lain adalah struktur indentasi dan *syntax* yang simpel membuat bahasa ini menjadi (relatif) lebih mudah untuk dipelajari dan dibaca. Programmer "dipaksa" untuk mengikuti aturan ini, nilai lebihnya adalah program menjadi relatif seragam antara buatan satu programmer dengan programmer lain.

Perkenalan saya dengan Python terjadi ketika tahun 2011 dan semenjak itu berusaha untuk mempelajari seluk beluk Python sampai saat ini. Harapan saya semoga buku ini dapat dijadikan bahan belajar bagi siapa saja yang tertarik dengan dunia pemrograman utamanya Python.

Saat ini mungkin bahasa pemrograman Python tidak setenar bahasa-bahasa lain seperti Java maupun PHP sehingga dokumentasi yang ada (dalam bahasa Indonesia) juga tidak terlalu banyak. Untuk itu, penulis tergerak untuk membuat panduan singkat belajar bahasa Python ini demi membantu mereka yang tertarik belajar Python dan juga membantu agar Python lebih dikenal lagi di masyarakat.

Satu hal yang pasti bagi kalian yang ingin belajar bahasa pemrograman, jangan takut dengan ***error***, karena pesan kesalahan itu ada sengaja diciptakan

untuk memudahkan kita sebagai programmer dalam mengetahui di mana letak kesalahan tersebut. Tanpa ada fitur *error handling*, programmer pasti kesulitan mencari dimana letak kesalahannya. Dan berita baiknya adalah pesan *error* di Python sangat mudah untuk dibaca, sehingga programmer lebih mudah dalam mengetahui di mana letak kesalahannya. Satu tips lagi yang disarankan oleh beberapa dokumentasi Python yang pernah saya baca, jika Anda ingin mempelajari Python, seringlah berlatih dan bereksperimen di Python *interpreter*, dan sebagai bonus dari buku ini nanti akan ada satu bab tersendiri yang membahas Python *interpreter*, yakni menggunakan IPython.

Dan sebagai catatan, proyek buku tutorial Python ini bukanlah proyek sekali jadi, artinya pengembangan dan perbaikan akan terus dilakukan, untuk itu sebagai seorang pemula, saya mohon kritik dan saran demi perkembangan buku ini di masa depan, semoga bermanfaat!. [Contact Me via Email]

Ucapan Terima Kasih

Ucapan terima kasih yang pertama saya haturkan kepada Allah SWT, yang kedua kepada junjungan kami yang Mulia Nabi Muhammad SAW.

Selanjutnya kepada istri, dan anak-anak kami yang telah membantu memberi kami semangat untuk terus berkarya demi kemajuan bersama.

Bab 1

Tentang Python

1.1 Sejarah Singkat Python

Mengapa sejarah itu penting? Karena itu salah satu hal yang membedakan manusia dengan makhluk ciptaan Tuhan yang lain. Bagi Anda yang merasa bahwa bagian ini tidak begitu penting, Anda dapat langsung melewati bagian ini. Bagaimana pun juga dengan mengetahui sejarah, setidaknya kita mengetahui dan memahami dasar filosofi dan tujuan diciptakannya bahasa Python ini.

Pada akhir dekade 1980-an, seorang programmer berkewarganegaraan Belanda, bernama Guido van Rossum yang saat itu bekerja di CWI (Institut Riset Matematika dan Ilmu Komputer di Belanda) sedang menggarap proyek menggunakan bahasa *ABC* pada sebuah komputer yang menggunakan sistem operasi *Amoeba*.

Bahasa pemrograman *ABC* termasuk bahasa yang cocok digunakan untuk pemula dalam dunia pemrograman, karea mempunyai *syntax* yang sederhana, tidak membutuhkan deklarasi variabel *variable declaration*, dan menggunakan sistem *indent* untuk menandai blok *statement*. Namun ada beberapa kelemahan dari *ABC* (salah satunya sistem penanganan kesalahan *error handling* yang kurang begitu solid), yang kemudian menginspirasi van Rossum untuk mengembangkan bahasa pemrograman sendiri.

Kelebihan-kelebihan yang dimiliki oleh bahasa *ABC* diambil dan kekurangan-kekurangan yang ada dicoba untuk dicarikan jalan keluarnya, sehingga muncullah bahasa pemrograman *Python* yang mudah untuk dipelajari bagi pemula, dukungan untuk pengembangan lebih jauh dan cukup tangguh untuk menyelesaikan berbagai permasalahan dengan efisien. Jadi dapat dikatakan bahwa Python dikembangkan dari bahasa *ABC*.

Lepas dari CWI, van Rossum kemudian bekerja pada BeOpen labs, yang kemudian bersama dengan teman-temannya mulai intensif mengembangkan Python 2.0 yang dilengkapi dengan fitur *list comprehensions* yang diambil dari bahasa

pemrograman *Haskell*[1].

Python 2.0 juga ditandai dengan dikembangkannya sejumlah fitur-fitur lain yang kemudian menyebabkan bahasa ini mulai dikenal di dunia pengembangan, seperti misalnya *garbage collection*, penerapan pemrograman berorientasi objek (OOP) yang memang menjadi salah satu tujuan dasar penciptaan bahasa ini.

Capaian terbesar dari Python adalah ketika Python menjadi bahasa standar, yang ditandai dengan dibentuknya *Python Software Foundation* sebagai lembaga yang bertanggungjawab terhadap pengembangan dan pengambil keputusan terhadap semua fitur-fitur yang akan diterapkan dalam Python. Kelompok ini menggunakan standar yang dinamakan *Python Enhancement Proposal* sebagai jalan bagi komunitas untuk memberikan kritik dan saran terhadap pengembangan Python. Saran dan pengembangan yang diterima oleh lembaga ini kemudian menjadi dasar rilis Python 3.0.

Itulah sekelumit sejarah tentang Python, pertanyaan selanjutnya adalah mau dibawa kemana bahasa ini kedepannya? Saya rasa tidak ada seorang pun yang tahu, namun yang jelas Python saat ini berguna sebagai sarana belajar untuk memasuki dunia pemrograman dan sebagai alat untuk memecahkan berbagai permasalahan di dunia pengembangan aplikasi.

1.2 Mengapa Python?

Saat ini ada ratusan bahasa pemrograman yang bisa kita pilih, mulai dari bahasa **C** dan **C++**, **Ruby**, **C#**, **Lua**, **Java**, atau yang populer seperti **PHP**, dari sekian banyaknya bahasa tersebut, mana yang akan kita pilih? Pertanyaan tersebut tidaklah mudah untuk dijawab. Penulis sendiri awalnya belajar PHP, yang kemudian tertarik untuk belajar Python sekitar 2 tahun yang lalu. Menurut hemat penulis, Python cocok untuk pemula yang ingin mengetahui seperti apa sih dunia programming itu? Python juga pilihan yang cukup bagus untuk memecahkan berbagai macam persoalan pemrograman.

Belakangan ini, Python semakin meningkat popularitasnya karena sifatnya yang dapat digunakan secara lintas-platform, baik itu di Windows, Linux/UNIX, maupun Macintosh atau bahkan pada handphone. Selain itu, Python juga didukung oleh koleksi library yang sangat banyak, sehingga memudahkan programmer dalam membantu memecahkan masalah pemrograman sehari-hari. Dan yang terakhir, Python itu bersifat *open-source* dan tentu saja '*free*'.

Python adalah bahasa pemrograman yang dikembangkan oleh Guido van Rossum pada tahun 1990-an. Nama Python itu sendiri terinspirasi dari serial komedi TV yang sedang "in" di kala itu, dan bukan berasal dari jenis ular (Python) seperti yang selama ini disangkakan oleh banyak orang.

Karena sifatnya yang mudah dipelajari dan aturan syntax yang sederhana, Python cocok digunakan untuk aplikasi-aplikasi dengan tenggat waktu yang pendek.

Kelebihan lain dari Python adalah mudah dibaca. Semakin mudah sebuah kode dipelajari, semakin mudah pula proses untuk men-*debug*, me-*maintain*, dan

[1] lebih jauh mengenai fitur *list comprehensions* akan dibahas pada sub-bab tersendiri

melakukan perubahan di masa yang akan datang. Salah satu yang membedakan Python dengan bahasa lain adalah masalah indentasi. Memang terasa agak aneh, tapi aturan ini menyebabkan kode Python sangat mudah untuk dibaca dan dipahami.

Kelebihan lain dari Python adalah filosofi "*batteries included*" artinya sekali Anda melakukan instalasi Python, Anda akan mendapatkan berbagai macam *library* yang siap untuk digunakan, mulai dari modul untuk menangani email, halaman web, basisdata, sistem operasi, pengembangan aplikasi GUI, dan lain sebagainya.

Itulah sebabnya mengapa perusahaan besar seperti Google, Rackspace, NASA dan masih banyak lagi lebih condong pada Python dalam pengembangan aplikasi yang mereka pakai.

Dibalik keunggulan-keunggulan tadi, Python juga menyimpan beberapa kelemahan yang mungkin tidak cocok untuk dipilih sebagai bahasa pemrograman untuk pengembangan aplikasi Anda, diantaranya, Python bukanlah bahasa yang tercepat, jika dibandingkan dengan bahasa C misalnya. Namun masalah kecepatan lambat laun tidak akan menjadi masalah karena hardware sekarang semakin murah dan murah saja, sehingga rata-rata orang mampu membeli hardware dengan spesifikasi yang cepat.

Selain itu, dalam hal jumlah pustaka (*library*) yang dimiliki, Python juga bukanlah bahasa pemrograman yang memiliki pustaka terbanyak, kenyataannya masih kalah dibanding C, Java dan Perl.

Kekurangan lain, Python tidak melakukan cek tipe variabel pada saat *compile*.

Bab 2

Instalasi Python

2.1 Lisensi Python

2.2 Mendapatkan Python

2.3 Instalasi Python

2.3.1 Instalalasi Modul Python

2.4 Komunitas Python

Bab 3

Pengenalan Singkat Python

Bab ini akan membawa Anda untuk mengenal syntax, semantic, kemampun dan filosofi dibalik bahasa pemrograman Python.

3.1 Python secara singkat

Python memiliki beberapa tipe data *built-in*, seperti *integer*, *float*, *complex number*, *string*, *list*, *tuple*, *dictionary*, dan *file object*. Kesemuanya ini dapat dimanipulasi menggunakan operator, fungsi-fungsi bawaan, pustaka fungsi-fungsi, atau tipe data yang memiliki metode tersendiri.

Programmer Python juga dapat mendefinisikan sendiri sebuah "*Class*" dan kemudian meng-*instantiate* sendiri *instance* dari sebuah *Class* (lebih lanjut akan dipelajari pada bagian OOP).

Python juga menyediakan struktur aliran kendali (*control flow*) seperti `if elif else`, dan juga `while` dan `for` *loops*. Programmer juga dapat dengan mudah melakukan *argument passing* dengan berbagai cara. Eksepsi (*Exception*) atau *error* juga dapat dimunculkan dengan *statement* `raise`, yang kemudian bisa dengan mudah dimanipulasi menggunakan `try-except-else`. Variabel yang ada dalam Python tidak harus dideklarasikan terlebih dahulu, dan dapat berisi berbagai tipe data, obyek, fungsi maupun modul.

3.1.1 Memulai Python Interpreter

Cara termudah mempelajari variabel dan ekpresi di Python adalah dengan melakukan eksperimen di Python *interpreter*. Anda dapat memulai Python *interpreter* dengan mengetikkan perintah berikut di terminal / console Anda (Linux dan Mac):

```
$ python
```

Untuk sistem operasi Windows, Anda dapat terlebih dahulu membaca cara instalasi Python. Apabila sudah berhasil, Anda akan dihadapkan pada tampilan yang kurang lebih bentuknya seperti ini:

```
Python 2.7.3 (default, Apr 20 2012, 22:44:07)
[GCC 4.6.3] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Tanda `>>>` menunjukkan *prompt* atau tanda bahwa Anda dipersilakan mengetikkan perintah terhadap Python *interpreter* tersebut. Sekarang mari kita mulai bermain-main dengan Python *interpreter* dengan melakukan operasi kalkulator sederhana:

```
>>> 2 + 2
4
>>> 40 * 2
80
>>> 4 - 3
1
>>> 5 / 2
2
```

Tidak ada yang istimewa dari operasi aritmatika sederhana ini, kecuali mungkin operasi pembagian di mana seharusnya kita dapatkan hasil `2.5`, tapi Python justru memberikan hasil `2`, kenapa?

Hal ini terkait dengan 2 hal, pertama versi Python yang saya gunakan adalah 2.x, dan yang kedua terkait dengan tipe data yang saya gunakan dalam operasi pembagian tersebut. Angka `5` dan `2` adalah `integer` sehingga Python juga akan memberikan hasil berupa data bertipe `integer`. Sekarang mari kita coba jika menggunakan data `float`:

```
>>> 5.0 / 2.0
2.5
```

Python sekarang tahu bahwa kita menginginkan data `float`, sehingga hasil yang diberikan persis seperti apa yang kita inginkan (`2.5`). Atau jika Anda menginginkan Python 2.x berperilaku sama dengan versi 3.x untuk masalah pembagian (*division*), Anda dapat mengaktifkan modul `__future__.division` dengan cara:

```
>>> from __future__ import division
>>> 5 / 2
2.5
```

Jika Anda menggunakan Python 3.x, secara *default* operasi di atas akan menghasilkan nilai `2.5`.

```
# Python 3.x
>>> 5 / 2
2.5
```

Berikutnya adalah belajar mencetak hasil output ke layar menggunakan perintah `print`:

```
>>> print 'Hello world!'
Hello world!
```

Atau jika Anda menggunakan Python versi 3.x:

```
>>> print('Hello world!')
Hello world!
```

Pada Python 2.x, `print` merupakan pernyataan (*statement*), di Python 3.x berubah menjadi fungsi *built-in*.

Mungkin pertanyaan Anda sekarang adalah Python versi berapa yang seharusnya saya pakai? Memang ada perbedaan yang sangat mendasar antara Python 2.x dengan 3.x, namun kenyataan sampai saat ini sebagian besar `library` masih menggunakan versi 2.x dan Python masih memberikan dukungan resmi pada versi ini sampai nanti proses migrasi selesai. Untuk itu, kami menyarankan menggunakan versi 2.x terlebih dahulu (secara *default* sudah ter-install pada Ubuntu 12.04). Jika Anda ingin mengetahui versi Python yang sekarang Anda gunakan, dapat diketikkan perintah berikut pada terminal:

```
$ python -V
Python 2.7.3
```

3.1.2 Penanda dan Operator (*Identifiers and Operators*)

Tanda `=` digunakan untuk memberikan nilai pada sebuah variabel, hanya sebagai nama untuk sebuah objek, tidak terkait dengan *memory location* seperti di bahasa **C**. Python tidak mengenal tipe (*loosely typed*, variabel yang semula *integer* di pagi hari, dapat berubah menjadi *string* di sore hari.) Python tidak mengenal deklarasi variabel, tetapi kita tidak dapat mengakses variabel sampai kita memberikan nilai pada variabel tersebut. Jika Anda mencoba akses variabel yang belum ada nilainya, Python akan memberikan komplain seperti berikut:

```
>>> a  # kita belum beri nilai ke a
Traceback (most recent call last):
    File "<stdin>", line 1, in <module>
NameError: name 'a' is not defined
```

Perlu diperhatikan juga bahwa sistem penamaan variabel di Python bersifat *case-sensitive*, artinya `hewan` tidak sama dengan `Hewan`, dan juga tidak sama dengan `heWan`. Seperti juga pada banyak bahasa pemrograman lain, nama variabel pada Python juga harus diawali dengan *letter* (A-Z atau a-z) atau *underscore* (_), dan diikuti dengan huruf, angka, dan *underscore*. Tidak ada batasan untuk panjang variabel, namun disarankan sesuai kebutuhan dan yang lebih penting lagi cukup menjelaskan nilai dari variabel tersebut.

```
wordCount               # valid
word_count              # valid
word_count_1            # valid
_word_count             # valid
_2                      # valid tapi tidak disarankan
8Word                   # invalid, dimulai dengan angka
word_coun't             # invalid, ada tanda petik
```

Ketika Anda berada di Python *interpreter*, tanda *underscore* mempunyai makna khusus, yakni merekam output dari ekspresi sebelumnya, perhatikan contoh berikut:

```
>>> 'Hello'
'Hello'
>>> _
'Hello'
>>> 5 + 2
7
>>> _ + 3
10
>>> _ * 10
100
```

Selain aturan penamaan variabel seperti di atas, Anda juga tidak dapat menggunakan kata-kata berikut ini, karena kata berikut ini merupakan kata yang sudah 'dipesan' oleh Python dan memiliki makna khusus:

```
and         del         for         is          raise
assert      elif        from        lambda      return
break       else        global      not         try
class       except      if          or          while
continue    exec        import      pass        def
finally     in          print
```

Python menggunakan tanda-tanda berikut sebagai operator (pemaknaan dilakukan sambil jalan):

```
-     !=    %     &     *     **    /     ^     |     ~
+     <     <<    <=    <>    ==    >     >=    >>
```

3.2 Tipe data Builtin

Python memiliki beberapa tipe data *built-in*, mulai dari *scalar* seperti angka (*numbers* dan *Booleans*, sampai yang kompleks seperti *list*, *dictionary*, dan *file*.

Tipe	Contoh
Integers	`1, -3, 42, 355, 88888, -7777777`
Floats	`3.0, 31e12, -6e-4`
Complex numbers	`3 + 2j, -4-2j, 4.2 + 6.3j`
Booleans	`True, False`

Tabel 3.1: Tipe Data Angka / Numbers

3.2.1 Numbers

Ada empat tipe data angka, yakni *integers*, *floats*, *complex numbers* dan *Booleans*:

Anda dapat memulai berlatih dengan tipe data ini dengan melakukan operasi aritmatika sederhana (kalkulator) seperti `+` (penjumlahan), `-` (pengurangan), `*` (perkalian), `/` (pembagian), `**` (pemangkatan), dan `%` (modulus / nilai sisa pembagian).

Berikut ini contoh untuk operasi aritmatika untuk *integers*:

```
>>> x = 5 + 2 - 3 * 2
>>> x
1
>>> 5 / 2  # menjadi 2.5 jika menggunakan Python 3.x
2
>>> 5 // 2
2
>>> 5 % 2
1
>>> 2 ** 8
256
>>> 1001 ** 3
1003003001
>>> 5 + 4 * 6
29
```

Perhatikan operasi terakhir menghasilkan angka `29` (bukan `56`), hal ini karena sesuai dengan aturan baru operasi aritmatika, di mana operasi perkalian akan dieksekusi lebih dulu. Apabila yang Anda inginkan `56`, maka Anda dapat memberi tahu Python *interpreter* dengan memberikan tanda kurung, seperti pada contoh berikut:

```
>>> (5 + 4) * 6
54
```

Operasi pembagian integer (`/`) menghasilkan angka dengan tipe *float* (baru di Python 3.x). Berikut beberapa contoh operasi aritmatika untuk tipe data *float*:

```
>>> x = 4.3 ** 2.4
>>> x
33.137847377716483
>>> 3.5e30 * 2.77e45
9.6950000000000002e+75
>>> 100000001.0 ** 3
1.0000000300000003e+24
```

Contoh berikut adalah untuk *complex numbers*:

```
>>> (3+2j) ** (2+3j)
(0.68176651908903363-2.1207457766159625j)
>>> x = (3+2j) * (4+9j)
>>> x
(-6+35j)
>>> x.real
-6.0
>>> x.imag
35.0
```

Complex numbers terdiri dari elemen riil dan elemen imajiner, yang diawali dengan huruf "`j`". Pada contoh di atas, variabel `x` berisi *complex number*. Untuk mendapatkan angka riil dari variabel `x`, Anda dapat mengetahuinya dengan cara menambahkan `x.real`. Selain beberapa operasi aritmatika di atas, ada juga beberapa fungsi-fungsi *built-in* Python yang dapat digunakan untuk melakukan operasi manipulasi pada angka. Ada juga modul `cmath` (yang berisi fungsi untuk manipulasi *complex numbers*, dan *library module* `math`, yang berguna untuk manipulasi ke 3 jenis angka lainnya. Berikut beberapa contoh penggunaan *library* `math`:

```
>>> round(3.49)
3
>>> import math
>>> math.ceil(3.49)
4
```

Contoh lain manipulasi pada jenis data *Booleans*:

```
>>> x = False
>>> x
False
>>> not x
True
>>> y = True * 2
>>> y
2
```

Pada contoh terlihat, bahwa selain `True` dan `False`, *Booleans* berperilaku seperti angka 1 (`True`) dan 0 (`False`).

Anda juga dapat menghitung panjang sebuah angka (digit) dengan cara merubah angka ke dalam tipe *string*, kemudian menerapkan fungsi `len`:

```
>>> len(str(2 ** 1000))
302
```

Python juga memiliki beberapa *module* bawaan yang berhubungan dengan tipe data *Angka* ini, misalnya *module* `math` dan `random`:

```
>>> import math
>>> math.pi
3.1415926535897931
>>> math.sqrt(81)
9.0
>>> import random
>>> random.random()
0.89102076208372638
>>> random.choice([1, 2, 3, 4])
2
```

3.2.2 Lists

Berikut contoh tipe data list dari Python:

```
[]
[1]
[1, 2, 3, 4, 5, 6, 7, 8, 12]
[1, "two", 3L, 4.0, ["a", "b"], (5, 6)]
```

List dapat berisi angka maupun campuran jenis data lain, seperti *string*, *tuples*, *list*, *dictionaries*, *file objects*, dan berbagai tipe angka. Sebuah list dapat di-indeks sesuai urutan dari depan atau dari belakang. Anda juga dapat melakukan subsegment atau lebih dikenal *slicing*, menggunakan notasi *slice*. Perhatikan contoh berikut:

```
>>> x = ["first", "second", "third", "fourth"]
>>> x[0]
'first'
>>> x[2]
'third'
>>> x[-1]
'fourth'
>>> x[-2]
'third'
>>> x[1:-1]
```

```
['second', 'third']
>>> x[0:3]
['first', 'second', 'third']
>>> x[-2:-1]
['third']
>>> x[:3]
['first', 'second', 'third']
>>> x[-2:]
['third', 'fourth']
>>> x[4]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list index out of range
```

Nomer indeks positif memulai pembelahan (*slicing*) dari sebelah kiri, sedangkan nomer indeks negatif memulai pembelahan dari sebelah kanan. Pembelahan menggunakan `[m:n]`, di mana `m` merupakan titik mulai, sampai dengan (tidak termasuk) `n`. Operasi `[:n]` berarti titik mulai dari awal (`0`), sampai dengan indeks ke `n`, dan operasi `[m:]` berarti titik mulai dari `m` sampai dengan akhir nomer indeks.

Apa yang terjadi ketika kita mencoba memanggil elemen yang tidak ada? Perhatikan pada baris terakhir, disitu terlihat ada peringatan *exception* dari Python yang intinya elemen yang kita cari (dalam hal ini elemen dengan indeks nomor 4) tidak ada dalam variabel `x` kita.

x=	[	"first" ,	"second" ,	"third" ,	"fourth"	]
Indeks Positif		0	1	2	3	
Indeks Negatif		-4	-3	-2	-1	

Tabel 3.2: Sistem Indeks List

Selain operasi pembelahan (*slicing*), dapat juga dilakukan penambahan, penghapusan, penggantian elemen dalam sebuah list, atau dapat juga membuat list baru dari list yang sudah ada. Untuk lebih jelasnya silakan lihat contoh berikut:

```
>>> x = [1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> x[1] = "two"
>>> x[8:9] = []
>>> x
[1, 'two', 3, 4, 5, 6, 7, 8]
>>> x[5:7] = [6.0, 6.5, 7.0]
>>> x
[1, 'two', 3, 4, 5, 6.0, 6.5, 7.0, 8]
>>> x[5:]
[6.0, 6.5, 7.0, 8]
```

Selain operasi di atas, list juga dapat dimanipulasi menggunakan *fungsi built-in* (`len, min` dan `max`), beberapa operator seperti (`in`, `+` dan `*`), serta beberapa *list method*, seperti `append, count, extend, index, insert, pop, remove, reverse` dan `sort`. Untuk lebih jelasnya perhatikan contoh berikut:

```
>>> x = [1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> len(x)
9
>>> [-1, 0] + x
[-1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> x.reverse()
>>> x
[9, 8, 7, 6, 5, 4, 3, 2, 1]
```

Perlu dicatat bahwa operator `+` dan `*` menciptakan list baru, tanpa merubah list semula (dalam hal ini list yang ada di variabel `x`).

Untuk mengetahui secara lengkap, metode / fungsi *built-in* sebuah *list*, dapat dilihat dengan cara mengetikkan perintah `dir(list)` pada Python *interpreter*.

```
>>> l = [1, 2, 3]
>>> dir(l)  # atau bisa juga dir(list)
['__add__', '__class__', '__contains__',
'__delattr__', '__delitem__', '__delslice__',
'__doc__', '__eq__', '__format__', '__ge__',
'__getattribute__', '__getitem__', '__getslice__',
'__gt__', '__hash__', '__iadd__', '__imul__',
'__init__', '__iter__', '__le__', '__len__',
'__lt__', '__mul__', '__ne__', '__new__',
'__reduce__', '__reduce_ex__', '__repr__',
'__reversed__', '__rmul__', '__setattr__',
'__setitem__', '__setslice__', '__sizeof__',
'__str__', '__subclasshook__', 'append', 'count',
'extend', 'index', 'insert', 'pop', 'remove',
'reverse', 'sort']
```

Sebagai tahap awal, Anda dapat mengesampingkan metode yang diawali dengan tanda _ dan mencoba mencari tahu apa kegunaan dari masing-masing fungsi tersebut. Inilah salah satu kelebihan Python, dokumentasi dapat langsung diakses dari *interpreter*!. Sebagai contoh, Anda ingin mengetahui apa kegunaan fungsi `append` pada *list*:

```
>>> help(l.append)
Help on built-in function append:

append(...)
    L.append(object) -- append object to end
(END)
```

Dan bantuan pun datang, sekarang kita tahu bahwa fungsi `append` adalah untuk *append* sebuah objek di bagian akhir *data sequence* (data berurut). Anda dapat keluar dari *mode help* dengan mengetikkan `q`.

3.2.3 Tuples

Tipe data *tuples* mirip dengan *list*, satu-satunya yang membedakan adalah bahwa *tuple* sifatnya *immutable* – yakni tidak dapat dimodifikasi setelah selesai diciptakan. Operator-operator yang ada pada *list* seperti `in`, `+`, `*`, dan fungsi-fungsi *built-in* seperti `len`, `max` dan `min`, juga dapat diterapkan pada *tuple*. Sistem indeks elemen dan notasi pembelahan (*slice notation*) juga berlaku pada *tuple*, tetapi tidak dapat menambahkan, menghilangkan atau pun mengganti elemen. *Tuple*mengenal dua macam metode, yakni `count` dan `index`. Tujuan utama penciptaan *tuple* adalah sebagai *keys* untuk bentuk data *dictionaries*. *Tuple* juga sangat membantu jika Anda menginginkan bentuk data yang tidak dapat dimodifikasi.

Perhatikan contoh berikut:

```
()
(1,)
(1, 2, 3, 4, 5, 6, 7, 8, 12)
(1, "two", 3L, 4.0, ["a", "b"], (5, 6))
```

Berbeda dengan list, untuk *(*tuple) dengan elemen tunggal, kita wajib memberi koma di belakang elemen tunggal tersebut, atau Python akan mengenalinya sebagai data dengan tipe yang lain. Perhatikan contoh berikut:

```
>>> angka = (1)
>>> type(angka)
<type 'int'>
>>> nama = ('sopier')
>>> type(nama)
<type 'str'>
>>> angka_tuple = (1,)
>>> type(angka_tuple)
<type 'tuple'>
```

Seperti juga *list*, elemen dalam *tuple* dapat berisi satu atau lebih macam tipe data, seperti *strings*, *tuples*, *dictionaries*, *fungsi*, *objek file*, dan berbagai macam tipe angka.

Sebuah *list* dapat dengan mudah dirubah menjadi *tuple* dengan menggunakan fungsi *built-in* `tuple`. Perhatikan contoh berikut:

```
>>> x = [1, 2, 3, 4]
>>> tuple(x)
(1, 2, 3, 4)
```

Dan sebaliknya juga, sebuah *tuple* dapat dengan mudah dirubah ke dalam bentuk *list*, menggunakan fungsi *built-in* `list`:

```
>>> x = (1, 2, 3, 4)
>>> list(x)
[1, 2, 3, 4]
```

3.2.4 Strings

Pemrosesan tipe data *string* merupakan salah satu kelebihan Python. Ada beberapa cara untuk mengekspresikan sebuah *string* dalam Python:

```
"A string in double quotes can contain 'single quote' characters."
'A string in single quotes can contain "double quote" characters.'
'''\This string starts with a tab and ends with a newline character.\n'''
"""This is a triple double quoted string, the only kind that can
contain real newlines."""
```

String diberi batas tanda (*delimited*) dapat dengan petik tunggal (`' '`), petik ganda (`" "`), petik 3 tunggal (`''' '''`), atau petik tiga ganda (`""" """`), dan dapat berisi penanda tab (`\t`) dan penandan baris baru (*newline*) `\n`.

Seperti *tuple*, *string* juga bersifat *immutable* (tidak dapat diubah). Apabila kita ingin memanipulasi sebuah *string*, maka kita harus menciptakan variabel baru untuk menyimpan hasil modifikasi tersebut.

Perhatikan contoh berikut:

```
>>> nama = 'kholid fuadi'
>>> nama[0]
'k'
>>> nama[0] = 'K'  # akan muncul pesan error
Traceback (most recent call last):
...
TypeError: 'str' object does not support item assignment
>>> nama_baru = 'K' + nama[1:]
>>> nama_baru
'Kholid fuadi'
```

String juga mengenal operasi `in`, `+`, dan `*`, serta fungsi *built-in* seperti `len`, `max`, dan `min` seperti halnya pada *lists* dan *tuples*. Sistem indeks dan notasi pembelahan juga berlaku pada *string*, namun sekali lagi tidak dapat digunakan untuk merubah variabel *string* yang sudah dideklarasikan sebelumnya.

Strings juga mengenal beberapa metode seperti `split` dan `replace`, perhatikan contoh berikut:

```
>>> x = 'belajar bahasa pemrograman Python'
>>> x.split()
['belajar', 'bahasa', 'pemrograman', 'Python']
>>> x.replace('belajar', 'mempelajari')
'mempelajari bahasa pemrograman Python'
```

Ada lagi fungsi `print` yang digunakan untuk mencetak output sebuah *string*.

```
>>> print(x) # Python 3.0 atau
'belajar bahasa pemrograman Python'
>>> print x # Python 2.x
'belajar bahasa pemrograman Python'
```

Anda juga dapat mengganti tipe data lain ke dalam tipe *string*.

```
>>> nilai = 9 # tipe data integer
>>> type(nilai)
<type 'int'>
>>> string_nilai = str(nilai)
>>> type(string_nilai)
<type 'str'>
```

Satu hal lagi terkait *string* yang menarik dan akan membantu sekali ketika bekerja dengan tipe data *string* di Python adalah fitur *string formatting*. Perhatikan contoh berikut:

```
>>> print '{0} versi {1}'.format('Python', 2.6)
Python versi 2.6
>>> print '%s versi %.1f' % ('Python', 2.6)
Python versi 2.6
```

Lebih lanjut mengenai fitur *string formatting* ini dapat Anda pelajari sendiri pada dokumentasi resmi pada situs resmi Python[1].

3.2.5 Dictionaries

Tipe data *dictionary* pada Python tak lain adalah tipe data *associative array* seperti pada bahasa pemrograman yang lain, memiliki pasangan *key* dan *value*. Kita dapat menghitung jumlah pasangan *key* dan *value* menggunakan fungsi *built-in* `len`, atau kita juga dapat menerapkan fungsi `del` untuk menghapus pasangan *key* dan *value*, beberapa fungsi lain yang bisa diterapkan pada *dictionary* diantaranya `clear, copy, get, has_key, items, keys, update` dan `values`. Berikut beberapa contoh dari terapan fungsi-fungsi tersebut:

```
>>> x = {1: "one", 2: "two"}
>>> x
{1: 'one', 2: 'two'}
>>> x["first"] = "one"
>>> x
{1: 'one', 2: 'two', 'first': 'one'}
>>> x[("Delorme", "Ryan", 1995)] = (1, 2, 3)
>>> x
```

[1] http://www.python.org

```
{1: 'one', 2: 'two', 'first': 'one', ("Delorme", "Ryan", 1995): (1, 2, 3)}
>>> x.keys()
[1, 2, ('Delorme', 'Ryan', 1995), 'first']
>>> x[1]
'one'
>>> x.get(1, "not available")
'one'
>>> x.get(4, "not available")
'not available'
```

Beberapa hal yang perlu dicatat dari tipe data *dictionaries* adalah bahwa *keys* harus berbentuk *immutable*, artinya kita tidak dapat menggunakan *list* dan *dictionary* sebagai *key* sebuah *dictionary*. Untuk *values*, semua tipe data dapat dimasukkan, ini berarti termasuk *list* dan *dictionary* itu sendiri.

Fungsi `get` akan mencari *value* dari parameter pertama yang kita masukkan, jika tidak ditemukan, maka Python akan memberikan *return* `None` atau kita dapat mendefinisikan sendiri hasil kembalian, dalam contoh di atas adalah `not available` (karena *key* `4` tidak ada dalam variabel `x`).

3.2.6 Sets

Sets adalah bentuk koleksi data yang tidak berurutan, di mana tiap-tiap data tersebut pasti unik (berbeda), tidak mungkin terdapat elemen yang nilainya sama dalam tipe data *set*. *Set* mirip dengan *key* dalam *dictionary*, tanpa adanya *value*. Perhatikan contoh berikut:

```
>>> x = set([1, 2, 3, 1, 3, 5])
>>> x
{1, 2, 3, 5}  # semua unik!
>>> 1 in x
True
>>> 4 in x
False
```

Anda dapat membuat *set* dengan menggunakan `set` terhadap sekelompok data, seperti *list*. Ketika data dirubah menjadi tipe *set*, maka data yang duplikat secara otomatis akan dihilangkan. Dalam contoh ada *keyword* `in` yang berguna untuk melakukan cek ada tidaknya data tersebut dalam sekelompok data (*membership checking*).

3.2.7 File Objects

Python juga memiliki *file object*, perhatikan contoh berikut:

```
>>> f = open("myfile", "w")  # mode w, berarti write
>>> f.write("Baris pertama dengan karakter newline\n")
38  # hanya akan keluar di Python versi 3.x
```

```
>>> f.write("Baris kedua, akan masuk kedalam file juga\n")
42
>>> f.close()
>>> f.open("myfile", "r")  # mode r berarti read
>>> line1 = f.readline()
>>> line2 = f.readline()
>>> f.close()
>>> print(line1, line2)
Baris pertama dengan karakter newline
Baris kedua, akan masuk kedalam file juga
>>> import os
>>> print(os.getcwd())
'/tmp'
>>> os.chdir(os.path.join("home", "banthink", "Dropbox", "python_book"))
>>> os.getcwd()
'/home/banthink/Dropbox/python_book'
>>> filename = os.path.join("tmp", "myfile")
>>> print(filename)
'/tmp/myfile'
>>> f = open(filename, "r")
>>> print(f.readline())
Baris pertama dengan karakter newline
>>> f.close()
```

Pernyataan (*statement*) `open` berarti menciptakan sebuah objek file. Parameter `w` berarti kita sedang bekerja dalam modus *write*. Setelah memasukkan 2 baris kalimat, kita menutup dengan pernyataan `close`, kemudian kita buka kembali *file* tersebut, namun kali ini dalam modus `r`, atau hanya *read* saja.

Modul `os` mempunyai banyak fungsi, diantaranya untuk mengetahui dalam direktori mana kita sekarang sedang bekerja `getcwd` (*get current working directory*), dan juga dapat digunakan untuk menyusun *path* menuju ke suatu *file* atau direktori tertentu.

3.2.8 Ringkasan

Berikut tabel ringkasan dari tipe data *built-in* dari Python

3.3 Control Flow Structures

Struktur aliran kendali atau *Control Flow Structures* merupakan salah satu hal yang esensial dalam sebuah bahasa pemrograman, dan Python menyediakan elemen ini secara lengkap. Mari kita lihat detilnya satu per satu.

Tipe Objek	Contoh
Numbers	`1234,3.1415,Decimal,Fraction`
Strings	`'spam', 'guido'`
Lists	`[1, [2, 'tiga', 4]`
Tuples	`(1, 'spam', 4, 'U')`
Dictionaries	`1: 'satu', 'rasa': 'enak'`
Sets	`set('abc'), 'a', 'b', 'c'`
File Objects	`myfile = open('filename', 'w')`

Tabel 3.3: Tipe Data *Built-in* Python

3.3.1 Booleans values dan expressions

Python memiliki beberapa cara mengekspresikan nilai *Boolean*, semua hal berikut bernilai `False`: konstanta *Boolean* `False` itu sendiri, `0, None`, nilai kosong seperti `[], ""`. Di sisi lain, konstanta `True` dan semua hal yang lain bernilai `True`.

Ekspresi perbandingan dapat dibentuk menggunakan operator perbandingan (`<, <=, ==, >=, !=, is, is not, in, not in`) dan operator logikal (`and, not, or`), yang pada akhirnya nilai ekspresi tersebut `True` atau `False`.

3.3.2 while loop

Bagi Anda yang sudah pernah belajar bahasa pemrograman tentu tidak asing dengan *while loop* ini. Struktur *while loop* dalam Python adalah sebagai berikut:

```
while condition:
    body
else:
    post-code
```

Contoh program:

```
1 x = 10
2 print 'Menghitung mundur dimulai...'
3 while x >= 0:
4     print x,
5     x -= 1       # sama dengan x = x - 1
6 else:
7     print
8     print 'selesai'
```

Output:

```
Menghitung mundur dimulai...
10 9 8 7 6 5 4 3 2 1 0
selesai
```

`condition` adalah ekspresi yang bertugas melakukan evaluasi apakah bernilai `True` atau `False`. Selama nilai dari ekspresi tersebut `True`, maka bagian `body` akan dijalankan terus-menerus, sebaliknya, jika bernilai `False`, maka `while` *loop* akan menjalankan bagian `post-code` dan skrip berhenti. Jika `condition` bernilai `False`, maka bagian `body` tidak akan berjalan sama sekali, kemudian langsung menjalankan bagian `post-code`. `Body` dan `post-code` terdiri dari satu atau *statement* yang dipisahkan oleh baris baru (*newline*) dengan level *indent* yang sama. Python *interpreter* menggunakan level *indent* ini untuk memisahkan satu bagian dengan bagian yang lain. Tidak dibutuhkan tanda pemisah lain, seperti kurung kurawal atau penanda lain.

Perhatikan bahwa bagian `else` dalam *while loop* sifatnya pilihan (opsional), dan seringkali tidak digunakan. Mengapa? Perhatikan kedua bentuk *looping* berikut:

```
while condition:
    body
else:
    post-code
```

Bandingkan dengan *while loop* berikut:

```
while condition:
    body
post-code
```

Bagian `post-code` pasti akan dijalankan oleh Python selama tidak ada perintah `break` pada bagian `body`, sehingga metode penulisan yang kedua (tanpa `else`) lebih disukai dan cenderung lebih mudah dipahami.

Bagaimana dengan pernyataan `break` dan `continue` pada *while loop* di Python? Seperti pada bahasa pemrograman lain, pernyataan `break` pada `body` akan menyebabkan *loop* terhenti, dan tidak akan menyentuh bagian `post-code` (jika ada bagian `else:`). Jika `continue`, maka bagian bawah dari `body` akan dilewati dan *loop* akan dimulai dari awal lagi.

3.3.3 if-elif-else statement (Pernyataan Kondisional)

Bentuk umum dari susunan `if-else` dalam Python adalah sebagai berikut:

```
if condition1:
    body1
elif condition2:
    body2
elif condition3:
    body3
.
.
.
elif condition(n-1):
```

```
11      body(n-1)
12  else:
13      body(n)
```

Berikut contoh program sederhana agar lebih mudah memahami struktur `if-else` dalam Python:

```
1  x = 1
2  y = 2
3
4  if x > y:
5      print 'x lebih besar dari y'
6  elif x < y:
7      print 'y lebih besar dari x'
8  else:
9      print 'x sama dengan y'
```

Dan outputnya tentu saja: `y lebih besar dari x`.

Inti dari `if-else` *looping* adalah selama kondisi `if` ataupun `elif` ataupun `else` bernilai `True`, maka `body` dalam bagian tersebut akan dieksekusi.

Perhatikan sekali lagi level indentasi yang digunakan, Python tidak memerlukan tanda `{}` maupun `[]` untuk menandai bagian `body`, melainkan cukup dengan membuat baris baru, dan buat level indent kedalam.

Tentu saja, Anda tidak perlu memakai struktur ini secara keseluruhan, kadang Anda hanya cukup membuat satu kondisi `if-else`, seperti:

```
1  if waktu == 'malam':
2      print 'tidur'
3  else:
4      print 'kerja'
```

Atau mungkin cuma satu `if` saja tanpa `else`:

```
1  if kondisi == 'ngantuk':
2      print 'istirahat'
3  # next code goes here
```

Perlu dicatat bahwa Python tidak memiliki pernyataan `case` atau `switch`, seperti yang (mungkin) pernah Anda temui dalam bahasa pemrograman lain.

3.3.4 for loop

Bagi Anda yang sudah pernah mempelajari bahasa pemrograman lain seperti PHP atau C, stuktur perulangan `for` pada Python agak sedikit berbeda. Pada bahasa C biasanya terdapat proses *incrementing* sebuah variabel dan kemudian melakukan testing terhadap variabel tersebut setiap kali melakukan iterasi (*iteration*).

Pada Python, perulangan `for` digunakan untuk melakukan iterasi terhadap objek yang bersifat *iterable*, seperti *string, list, tuple* atau objek yang lain selama dia terdiri dari urutan / elemen-elemen.

Perulangan `for` juga dapat diterapkan pada fungsi seperti `range`, atau fungsi khusus yang disebut *generator*.

Berikut bentuk struktur perulangan `for` pada Python:

```
1 for item in sequence:
2     body
3 else:
4     post-code
```

Keterangan: Bagian *body* akan dieksekusi oleh tiap-tiap elemen dalam sequence. Bagian *else*, seperti juga pada perulangan `while` sangat jarang ditemukan dalam dunia nyata. Pernyataan `break` dan `continue` melakukan hal yang sama dengan yang dilakukan oleh perulangan `while`.

Contoh program:

```
>>> for i in range(4):
...     print i ** 2
...
```

Output:

```
0
1
4
9
```

`range` adalah sebuah fungsi yang menghasilkan urutan indeks dalam bentuk `list`, dalam contoh di atas, `range(4)` berarti `[0, 1, 2, 3]`, ingat mulai selalu dari `0`. Anda dapat mencoba sendiri menggunakan Python *interpreter* yang ada di komputer Anda.

Proses selanjutnya Python akan melakukan perkalian kuadrat (`**`) terhadap masing-masing elemen tersebut kemudian mencetak kembali ke layar menggunakan perintah `print` (baris 2).

Mumpung masih bicara tentang `range`, ada beberapa fitur menarik dari fungsi tersebut yang bisa kita pakai, diantaranya argumen opsional `step`. Sebelum menuju ke contoh kode, mari kita biasakan menengok dokumentasi dari fungsi `range` itu sendiri. Di sini saya memakai *iPython interactive interpreter*.

```
>>> help(range)  # atau cukup <range?> di iPython
Type:       builtin_function_or_method
Base Class: <type 'builtin_function_or_method'>
String Form:<built-in function range>
Namespace:  Python builtin
Docstring:
range([start,] stop[, step]) -> list of integers

Return a list containing an arithmetic progression of integers.
range(i, j) returns [i, i+1, i+2, ..., j-1]; start (!) defaults to 0.
```

```
When step is given, it specifies the increment (or decrement).
For example, range(4) returns [0, 1, 2, 3].  The end point is omitted!
These are exactly the valid indices for a list of 4 elements.
```

Fungsi menerima 3 argumen, satu wajib (`stop`), dua lainnya opsional. Untuk argumen `start` sudah jelas (*default* mulai dari 0), yakni dari angka berapa dimulai. Untuk `step` ini adalah argumen yang digunakan untuk '*skip*' atau melompati berapa angka, perhatikan contoh berikut:

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

Terlihat bahwa nilai kembalian (*return*) dari fungsi `range` tersebut adalah angka genap ('lompat dua-dua'), menarik bukan? Ada satu trik lagi, agar kita bisa menghasilkan `range` yang urut dari angka paling belakang dulu (terbalik), perhatikan kode berikut:

```
>>> range(5, 0, -1)
[5, 4, 3, 2, 1]
```

Trik-trik seperti ini nantinya akan sering digunakan dalam dunia nyata, jadi silakan bereksperimen sebanyak dan sesering mungkin!

3.4 Fungsi

Fungsi merupakan bentuk dasar dari program di Python yang berguna untuk memaksimalkan penggunaan kode secara berulang-ulang (*code reuse*) dan menekan tingkat *code redundancy*. Fungsi juga berguna untuk memecah suatu persoalan yang kompleks menjadi bagian-bagian yang lebih kecil dengan harapan menjadi lebih mudah dipecahkan.

Disadari atau tidak, sebenarnya kita sudah menggunakan beberapa fungsi pada pembahasan sebelumnya, sebagai contoh fungsi `range`, fungsi ini dinamakan sebagai fungsi bawaan (*builtin function*) yang tersedia setiap pada setiap program Python. Selanjutnya yang akan dibahas di sini adalah bagaimana kita membuat fungsi baru dalam Python.

3.4.1 Definisi Fungsi

Secara singkat, fungsi didefinisikan sebagai kumpulan pernyataan (*set of statements*) yang dapat dijalankan secara berulang-ulang dalam sebuah program. Fungsi dapat memiliki parameter sebagai *input*, yang setiap kali dijalankan bisa memberikan nilai *return* yang berbeda-beda.

Syntax dasar penulisan sebuah fungsi pada Python:

```
def nama_fungsi(param1, param2, ...):
    body
```

Fungsi terdiri dari `def` sebagai *header* dan diikuti dengan *body* yang terdiri dari pernyataan-pernyataan (*statements*). Untuk memisahkan ini umumnya digunakan indentasi, atau jika fungsi pendek, pernyataan cukupn ditambahkan setelah tanda titik dua (`:`).

```
def func(param1, param2, ...): body
```

Jika diperhatikan, fungsi juga memiliki aturan indentasi yang memisahkan antara tubuh fungsi (*body of function*) dengan definisi fungsi (*function definition*).

Contoh fungsi penghitung nilai faktorial pada Python:

```
1 >>> def fakt(n):
2 ...     """Menghitung nilai faktorial dari n"""
3 ...     r = 1
4 ...     while n > 0:
5 ...         r = r * n
6 ...         n = n - 1
7 ...     return r
8 ...
```

Mari kita pelajari struktur dari fungsi tersebut. Baris 1 adalah nama dari fungsi, dalam hal ini `fakt`, di mana fungsi ini menerima satu nilai (`n`) sebagai *parameter*.

Baris 2 dinamakan sebagai string dokumentasi (*documentation string* atau *docstring*). *Docstring* adalah string yang menjelaskan perilaku dari fungsi dan parameter yang dibutuhkan. Apa bedanya dengan sistem komentar (*comment*)? Komentar menjelaskan sesuatu yang lebih spesifik, misalnya informasi mengenai bagaimana sistem kerja sebuah baris kode. *Docstring* merupakan baris pertama setelah definisi fungsi, dan biasanya menggunakan *triple quote* (`""" """`), memungkinkan untuk deskripsi yang terdiri lebih dari satu baris (*multiline*). Standar penulisan *docstring* adalah dengan menuliskan sinopsis pada baris pertama, diikuti dengan baris kosong pada baris kedua, baru setelah itu penjelasan lain terkait dengan perilaku dan parameter dari fungsi.

Meski bersifat opsional, Python sangat menyarankan setiap modul/class/-fungsi memiliki *docstring* tersebut, walaupun hanya satu baris (lihat bagian Coding Style. Anda dapat mengakses dokumentasi ini dengan cara:

```
>>> fakt.__doc__
'Menghitung nilai faktorial dari n'
```

Selanjutnya, baris 3 - 6, adalah pendeklarasian variabel `r` dan *looping* menggunakan `while`. Baris terakhir (7), berisi pernyataan `return` yang berarti nilai yang akan dihasilkan oleh fungsi tersebut ketika fungsi tersebut dipanggil (*call*), dalam hal ini nilai faktorial dari `n`.

Sekarang waktunya mencoba fungsi tersebut:

```
>>> fakt(4)
24
>>> fakt(5)
120
```

Yap, setidaknya menurut teori faktorial yang saya ketahui, hasil tersebut sudah benar, artinya fungsi `fakt` sudah berjalan sesuai kodratnya...

3.4.2 Memahami Parameter dalam Fungsi

Hampir sebagian besar fungsi membutuhkan parameter, dan tiap bahasa pemrograman memiliki spesifikasi dan aturan tertentu bagaimana sebuah parameter digunakan dalam fungsi. Aturan Python termasuk fleksibel dan menyediakan tiga pilihan bagaimana mendefinisikan parameter dalam sebuah fungsi. Berikut secara sekilas ketiga pilihan tersebut:

3.4.2.1 Parameter Posisional

Cara paling mudah melakukan *passing* parameter terhadap sebuah fungsi adalah dengan menggunakan posisi (sesuai urutan). Perhatikan contoh berikut:

```
>>> def perkalian(x, y):
...     return x * y
...
>>> perkalian(3, 7)
21
```

Perhatikan ketika kita memanggil fungsi `perkalian`, ada 2 parameter yang dipakai disini, `x` dan `y`. Sesuai dengan urutan parameter yang kita definisikan dalam fungsi, Python mengetahui bahwa parameter `x` bernilai 3 dan parameter `y` bernilai 7.

Perlu diketahui bahwa jika menggunakan metode ini, jumlah parameter yang ada dalam fungsi harus sesuai dengan jumlah parameter yang kita pakai ketika memanggil sebuah fungsi. Fungsi `perkalian` memiliki 2 buah parameter (`x` dan `y`). Bagaimana jika kita memanggil fungsi tersebut menggunakan cuma 1 parameter saja? Python akan komplain dan memunculkan eksepsi `TypeError` yang kurang lebih tampilannya sebagai berikut:

```
>>> perkalian(8)
Traceback (most recent call last):
    File "stdin", line 1, in <module>
TypeError: perkalian() takes exactly 2 positional arguments (1 given)
>>>
```

Untuk menghindari `TypeError`, salah satunya kita dapat menggunakan nilai *default* untuk parameter yang kita gunakan sebagai contoh:

```
def somefunc(arg1, arg2=default2, arg3=default3):
```

Jika diterapkan pada fungsi `perkalian` kita di atas:

```
>>> def perkalian(x, y=3):
...     return x * y
...
>>> perkalian(8, 3)
24
>>> perkalian(8)
24
```

3.4.2.2 Variable numbers of arguments

Perhatikan kode berikut:

```
>>> def somefunc(*a):
...     print a
...
>>> somefunc('a', 'b', 'c')
('a', 'b', 'c')
```

Penggunaan tanda *asterisk* pada argumen mengakibatkan fungsi akan memanggil semua argumen posisional kemudian Python merubah semua argumen tersebut ke dalam `tuple`.

Berbeda dengan *positional argument*, *keyword argument* ditandai dengan tanda *double asterisk* (`**`). Bagaimana cara Python melakukan ekstraksi terhadap keyword argument tersebut? Perhatikan contoh berikut:

```
>>> def example(x, y, **other):
...     print "x: {0}, y: {1}, keys in 'other': {2}".format(x, y,
...         other.keys())
...     other_total = 0
...     for k in other.keys():
...         other_total = other_total + other[k]
...     print "The total of values in 'other' is {0}".format(
...         other_total)
...
>>>
```

Pertama kita mendefinisikan fungsi bernama `example` yang menerima beberapa argumen (setidaknya 2 *positional argument* dan 1 *keyword argument*). Untuk memastikan bahwa semua argument diterima oleh fungsi tersebut, kita melakukan perintah *printing* sesuai dengan urutan argumen, dan khusus untuk argumen `other`, kita coba lihat *key* apa saja yang dimiliki oleh `other` dengan menerapkan fungsi `keys`. Keluaran dari fungsi `other.keys()` ini adalah tipe data *list* yang bersifat *iterable* sehingga bisa kita hitung jumlah totalnya dan dimasukkan dalam variabel `other_total`.

Pada dasarnya kita juga boleh mencampur teknik *argument passing* menjadi satu, namun hal ini kadang menjadi membingungkan, lihat contoh berikut:

```
>>> def func(a, b, *c, **d):
...     print a, b, c, d
...
>>> func(1, 2, 3, 4)
1 2 (3, 4) {}
>>> func(1, 2, 3, some=4)
1 2 (3,) {'some': 4}
```

Bagi yang tertarik memperdalam teknik-teknik *argument passing* secara lebih mendalam dapat merujuk dokumentasi resmi Python[2].

3.5 Exceptions

Exceptions atau eksepsi atau mungkin dalam bahasa Indonesia dapat dipadankan dengan kata 'pengecualian'. Fungsinya adalah untuk menangani kejadian yang tidak biasa (*unusual*) setelah program berjalan. Yang paling umum dijumpai adalah eksepsi untuk menangani *error* yang muncul selama program berjalan, misal pesan kesalahan file tidak dapat dibuka, dan lain sebagainya. Python sudah menyediakan fitur-fitur tersebut, dan kita dapat membuat eksepsi baru sesuai kebutuhan kita.

Konsep *exception* sebagai mekanisme penanganan kesalahan (*error handling*) sudah dikenal beberapa lama. C dan Perl sebagai bahasa *scripting* yang dikenal luas, ternyata tidak memiliki fitur tersebut, bahkan mereka yang sering bekerja dengan C++ kadang tidak terlalu familiar dengan konsep ini.

3.6 Membuat module

3.7 Object-oriented programming

3.8 Lain-lain

3.8.1 List Comprehensions

3.8.2 Regular Expression

3.8.3 Mastering IPython

This will be fun!

3.8.4 Coding Style

Sekarang Anda sudah siap untuk menulis kode Python ke dalam sebuah / lebih file, ini artinya sekarang saat yang tepat untuk membicarakan mengenai *coding*

[2] http://www.python.org

style. Ide besar dari tema ini adalah agar kode yang Anda tulis menjadi lebih mudah dibaca pengguna lain.

Python sendiri menyediakan sejumlah aturan (standar) yang dinamakan dengan PEP 8[3], yang sangat disarankan untuk dibaca bagi semua programmer Python, ke-8 aturan tersebut adalah:

- Gunakan indentasi 4 spasi, dan jangan gunakan *tabs*.
  4 spasi adalah ukuran yang cukup, tidak terlalu lebar sehingga lebih mudah dibaca.
- Batasi jumlah karakter dalam 1 baris maksimal 79 karakter.
  Hal ini akan membantu bagi yang menggunakan *display* kecil dan memungkinkan melakukan *split screen* sehingga beberapa file dapat ditampilkan secara menyeluruh dalam satu *display*.
- Gunakan baris kosong untuk memisahkan *function* dengan *class*, dan memisahkan definisi fungsi dengan *body*.
- Jika memungkinkan, taruh *comment* pada baris kode yang bersangkutan.
- Jangan lupakan *docstring*.
- Gunakan spasi antara *operators* dan setelah koma, kecuali untuk kode yang ada di dalam konstruksi kurung (*brackets*).
- Gunakan sistem penamaan yang konsisten terhadap fungsi dan klas Anda. Kesepakatan yang umum adalah menggunakan sistem `CamelCase` untuk klas dan `lower_case_with_underscores` untuk fungsi dan method. Selalu gunakan `self` sebagai nama dari argument pertama dari method.
- Jangan gunakan sistem *encoding* yang aneh-aneh jika kode Anda ditujukan untuk kalangan internasional. `Plain ASCII` adalah pilihan yang tepat untuk semua situasi.

3.9 Penutup

[3] http://www.python.org/dev/peps/pep-0008/