BUDI RAHARDJO

KODING PYTHON

Contents

Pengantar

Buku ini sebetulnya merupakan catatan pribadi saya dalam belajar pemrograman dengan menggunakan bahasa Python. Saya sudah mengenal Python sejak dari jaman dahulu kala, tetapi pada masa itu saya tidak terlalu tertarik karena saya lebih suka menggunakan bahasa Perl. Sampai sekarang sebetulnya saya masih suka menggunakan bahasa Perl, tetapi karena tuntutan zaman yang banyak membutuhkan pemrograman dengan menggunakan bahasa Python maka saya kembali mempelajari bahasa Python.

Buku ini lebih banyak menampilkan contoh-contoh yang saya gunakan untuk mengingat-ingat hal-hal yang pernah saya kerjakan atau untuk mencari ide ketika memecahkan masalah lain. Jadinya buku ini seperti sebuah *cookbook*. Semoga pendekatan seperti ini cocok juga untuk Anda.

Yang namanya catatan tentu saja sesuai dengan apa yang saya lakukan. Basis saya menggunakan Linux. Jadi ada kemungkinan contoh yang tidak persis sama. Demikian pula cara saya menggunakan (memprogram dengan) Python mungkin bukan cara yang paling sempurna, tetapi mengikuti cara saya. (Apapun itu.)

~~Ketika buku ini ditulis, versi Python yang paling stabil adalah versi 2.7 meskipun versi 3 juga sudah banyak digunakan orang. Ada banyak bagian di dalam buku ini yang dituliskan untuk Python versi 2.7 kemungkinan harus disesuaikan untuk versi 3. Namun secara prinsip mestinya sama.~~

Versi buku ini sudah menganjurkan untuk lebih condong ke versi 3 karena pengembangan versi 2 sudah mulai dihentikan. [Catatan: buku mulai ditulis akhir 2017.] Bagian-bagian yang sudah terlanjur ditulis dengan menggunakan versi 2, sedikit demi sedikit mulai dikonversikan ke versi 3.

Selamat menikmati versi 0.3.

Bandung, November 2020

Budi Rahardjo

1
Pendahuluan

Bahasa pemrograman Python mulai populer saat dikarenakan berbagai hal; mudah dipelajari, tersedia dan banyak *library*-nya. Nanti akan kita bahas beberapa library Python ini. Lengkapnya library ini juga yang menyebabkan Python dipergunakan di berbagai aplikasi. Berbagai sekolah (dan perguruan tinggi) bahkan mengajarkan Python sebagai pengantar pemrograman.

Bahasa Python merupakan sebuah *interpreted language* berbeda dengan bahasa C yang *compiled*. Pada bahasa yang *compiled*, kita memiliki kode sumber (*source code*) yang harus dirakit (*compile*) dahulu sampai menjadi kode mesin yang langsung dapat dieksekusi pada komputer yang bersangkutan. Ketika algoritma salah, maka kode sumber harus diperbaiki dahulu kemudian di-*compile*) sebelum dapat dijalankan. Prosesnya menjadi agak panjang. Sementara itu untuk bahasa yang *interpreted*, program langsung dieksekusi dari kode sumbernya (tanpa perlu proses kompilasi). Dahulu program yang *compiled* lebih cepat dalam eksekusinya karena tidak perlu menerjemahkan baris perbaris ketika dijalankan, namun sekarang perbedaannya sudah tipis.

Bahasa Python tersedia untuk berbagai sistem operasi; Windows, Mac OS, dan berbagai variasi dari UNIX (Linux, *BSD, dan seterusnya). Di dalam buku ini saya akan menggunakan contoh-contoh yang saya gunakan di komputer saya yang berbasis Linux Mint. Meskipun seharusnya semuanya kompatibel dengan berbagai sistem operasi, kemungkinan ada hal-hal yang agak berbeda. Jika hal itu terjadi, gunakan internet untuk mencari jawabannya.

1.1 Instalasi

Python dapat diperoleh secara gratis dari berbagai sumber. Sumber utamanya adalah di situs python.org. Untuk sementara ini bagian ini saya serahkan kepada Anda. Ada terlalu banyak perubahan sehingga bagian ini akan cepat kadaluwarsa. Untuk sistem berbasis sistem

operasi Microsoft Windows, biasanya instalasi Python menggungankan *Anaconda*. (Informasi mengenai ini juga dapat dilihat pada situs python.org.)

Untuk sistem operasi berbasis Linux dan Mac OS, Python sudah terpasang sebagai bawaan dari sistem operasinya. Jika Anda ingin memasang versi terbaru maka Anda harus memasangnya sendiri dengan mengunduh instalasinya di python.org. Atau, jika Python sudah terpasang di komputer Anda, maka Anda dapat melakukan *upgrade*.

Ada cara lain menggunakan Python adalah dengan menggunakan layanan Google Colabs, yang mana kita diberikan akses ke sebuah mesin virtual yang sudah terpasang Python. Untuk pendekatan ini kita tidak perlu memasang Python lagi. Lebih mudah untuk belajar dan bahkan untuk membuat prototipe. Buku ini akan diperbaharui dengan cara menggunakan Google Colabs ini.

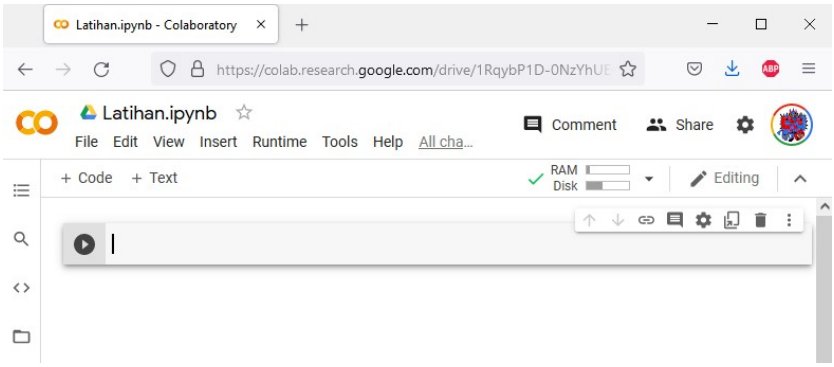

Figure 1.1: Tampilan Google Colab

Layanan Google Colab dapat diakses dengan menggunakan *browser* dan diarahkan ke alamat colab.research.google.com (atau gunakan search engine untuk mencapai domain tersebu). Setelah ditampilkan halaman depan, kita dapat membuka program baru dengan membuka *New Notebook* atau membukan kodingan lama kita yang sudah disimpan di akun Google Drive kita. Google colab menyediakan sebuah komputer virtual yang dapat kita gunakan untuk melakukan pemrograman dalam bahasa Python (yang dalam hal ini dapat kita pilih mau menggunakan versi 2 atau 3).

1.2 *Memulai*

Untuk memastikan Python berjalan, ketikkan "python" di terminal Linux Anda. (Bagi yang menggunakan Windows, hal ini dapat dilakukan dengan menggunakan CMD.exe.) Catatan, di sistem Linux, tanda “dollar” merupakan *prompt* dari *shell* Anda. Jangan diketikkan.

```
$ python
Python 2.7.12 (default, Nov 20 2017, 18:23:56)
[GCC 5.4.0 20160609] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Dari tampilan di atas dapat kita ketahui bahwa Python yang saya gunakan adalah versi 2.7.12[1]. Sekarang kita dapat memulai pemrograman Python dengan menuliskan program "hello world" (yang merupakan standar bagi belajar pemrograman). Ketikkan "print ..." (dan seterusnya seperti di bawah ini).

[1] Pada awal penulisan buku ini, versi bahasa Python Yang stabil digunakan adalah versi 2. Saat ini versi 3 merupakan versi yang dianggap sebagai versi stabil. Untuk itu buku ini sedikit demi sedikit akan diperbaharui dengan menggunakan Python versi 3.

```
print "Hello, world!"
Hello, world!
```

Python akan menampilkan apapun yang ada di antara tanda petik tersebut. Hore! Anda berhasil membuat program Python yang pertama.

Ada beberapa cara untuk menjalankan Python. Pada contoh di atas, kita menjalankannya secara langsung. Cara ini memang yang paling cepat, tetapi ada banyak hal yang harus kita lakukan secara manual. Sebagai contoh, jika kita ingin membuat sebuah *block*, maka kita harus mengetikkan sendiri empat spasi untuk membuatnya masuk. Jika ini tidak kita lakukan, maka dia akan "marah" dan menampilkan pesan. Berikut ini contoh sesi yang salah.

```
$ python3
Python 3.5.2 (default, Nov 23 2017, 16:37:01)
[GCC 5.4.0 20160609] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> for i in range(10):
... print(i)
  File "<stdin>", line 2
    print(i)
        ^
IndentationError: expected an indented block
>>>
```

Pada contoh di atas, kesalahan terjadi karena kita tidak memberikan spasi di depan perintah "print(i)". Seharusnya kita melakukan hal seperti ini. (Perhatikan spasi sebelum kata "print".)

```
$ python3
Python 3.5.2 (default, Nov 23 2017, 16:37:01)
[GCC 5.4.0 20160609] on linux
Type "help", "copyright", "credits" or "license" for more information.
```

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

Mari kita lanjutkan dengan membuat program yang lebih panjang. Program Python dapat disimpan di dalam sebuah berkas untuk kemudian dieksekusi belakangan. Buka editor kesukaan Anda dan ketikkan program hello world di atas di dalam editor Anda tersebut. Setelah itu simpan berkas tersebut dengan nama "hello.py". Biasanya berkas program Python ditandai dengan akhiran (extension) ".py".

Setelah berkas tersebut tersedia, maka kita dapat menjalankan Python dengan memberikan perintah python dan nama berkas tersebut. Kata "Hello world" akan ditampilkan.

```
$ python hello.py
Hello, world!
```

Cara lain yang dapat dilakukan adalah dengan menggunakan program IPython, yaitu interactive Python. Biasanya program IPython ini belum terpasang di sistem operasi bawaan komputer Anda. Tinggal unduh dari ipython.org. (Perhatikan IPython ini sudah mengenal *syntax* dari Python sehingga ketika kita mengetikkan *for loop* maka dia akan memberikan spasi 4 buah sebagai *indentation*.)

```
$ ipython3
In [1]: print("Hello World")
Hello World

In [2]: for i in range(10):
   ...:     print(i)
   ...:
0
1
2
3
```

```
4
5
6
7
8
9

In [3]:
```

Masih ada satu cara lain menjalankan program Python, yaitu dengan menggunakan Jupyter. Yang ini akan kita bahas secara lebih khusus.

Pada contoh-contoh di atas hasil print dicetak ke bawah. Bagaimana jika kita ingin hasil cetaknya tidak turun ke bawah atau tanpa *new-line*? Cara berikut ini - dengan menambahkan (end = " ") - dapat digunakan:

```
for n in range(10):
   print(n, end=" ")
```

Keluaran dari program di atas adalah cetakan yang ke kanan.

```
0 1 2 3 4 5 6 7 8 9
```

1.3 Shell atau IDE

Pemrograman Python dapat dilakukan dengan editor seperti yang sudah ditunjukkan di atas dan kemudian menjalankannya secara *command line* di shell, Cara lain yang lebih sering dilakukan orang adalah menggunakan sebuah Integrated Development Editor (IDE) untuk melakukan pemrogramannnya. Menjalankan kodenya dapat dikomandokan dari IDE atau dari shell juga.

Ada beberapa IDE untuk Python, seperti misalnya IDLE. Yang sekarang sedang naik daun adalah dengan menggunakan *Jupyter Notebook*. Yang menarik dengan pendekatan ini adalah kita melakukan dokumentasi dan pemrograman sekaligus di lingkungan tersebut. Google Colab pun sesungguhnya menggunakan konsep Jupyter Notebook ini tetapi membawanya ke lingkungan *cloud*. Nanti ini akan kita bahas dengan lebih lanjut.

1.4 Bahasa Python

Tentang bahasa Python itu sendiri akan diperdalam pada versi berikutnya. Sementara itu fitur tentang bahasa Python akan dibahas sambil berjalan. Pendekatan ini saya ambil untuk membuat buku

menjadi lebih menarik dan lebih singkat. Belajar seperlunya. Mari kita mulai.

Variabel di dalam Python langsung dapat digunakan tanpa melakukan deklarasi sebelumnya. Pada bahasa pemrograman seperti C, variabel harus dideklarasikan tipenya; apakah dia *integer* atau *string*. Contohnya di bawah ini.

```
a = 7
b = 5
print(a,b)
```

Pada contoh di atas, variabel *a* dan *b* dibuat dan langsung diisi dengan angka (7 dan 5) dalam kasus ini. Kemudian kedua variabel tersebut disampaikan sekaligus. Perhatikan bahwa dengan menggunakan tanda koma (,) nilai dari kedua variabel tersebut ditampilkan dengan spasi.

```
7 5
```

Berikut ini kita buat penampilan yang lebih "menarik" (untuk Python3).

```
a = 7
b = 5
c = a + b
print ("a = ", a)
print ("b = ", b)
print ("a+b = ", c)
```

Keluaran dari Python3 adalah seperti ini:

```
a = 7
b = 5
a+b = 12
```

Hal yang sangat berbeda dari bahasa Python dengan bahasa pemrograman lainnya adalah masalah *block* dari kode. Bahasa pemrograman C misalnya menggunakan tanda kurung kurawal "{" untuk menyatakan blok. Sementara itu Python menggunakan *indentation* untuk menyatakan satu blok. Lihat contoh di bawah ini.

```
for i in range(5):
    for j in range(3):
        print(i,j)
```

Disarankan untuk menggunakan spasi sebanyak empat (4) buah untuk *indentation* tersebut. (Ini membuat banyak perdebatan karena ada banyak orang yang menggunakan tab bukan spasi.)

Mari kita buat contoh-contoh lain. Apa keluaran program di bawah ini?

```
nama1 = "budi"
nama2 = "rahardjo"
nama3 = nama1 + nama2
print(nama3)
```

Perhatikan bahwa untuk variabel yang berjenis angka (integer) maka operator tambah (+) akan menambahkan angkanya. Sementara itu pada variabel yang berjenis *string*, operator (+) akan menyambungkan (*concatenate*) variabel *string* tersebut. Ini merupakan ciri dari sebuah bahasa yang berorientasi obyek (*object oriented*). Untuk tipe data yang lainnya, operator tambah (+) kemungkinan juga akan memiliki perilaku yang berbeda.

Mari kembali ke data yang berbentuk angka. Apa keluaran dari program di bawah ini?

```
a = 7
b = 5
c = a/b
print(c)
```

Keluarannya adalah nilai "1.4". Jenis atau tipe data dari "a" dan "c" adalah berbeda. Tipe data "a" adalah *integer*, sementara itu tipe data "c" adalah *floating point*.

```
>>> a=7
>>> type(a)
<class 'int'>
>>> b=5
>>> c=a/b
>>> type(c)
<class 'float'>
```

Bolehkah kita mencampurkan tipe data yang berbeda? Mari kita lanjutkan dengan contoh di atas.

```
d = a+c
print(d)
```

Keluaran dari kode di atas adalah "8.4". Artinya pencampuran tipe data yang berbeda (dalam hal ini adalah *integer* dan *float*) dapat dilakukan. Hasilnya adalah bilangan *float*. Hal ini menunjukkan bahwa bahasa Python dapat dibuat tidak *strict*. Bahasa pemrograman lain - misalnya Java (dan umumnya bahasa pemrograman yang berorientasi obyek) - sangat ketat dalam hal ini.

1.5 Kompleksitas

Pada bagian sebelumnya kebetulan sempat disinggung tentang *loop*, maka pada bagian ini pembahasannya sedikit melebar ke aspek

Kompleksitas (dari subuah algoritme). Ini memang lebih berbau aspek teoritis dari ilmu komputer.

Ketika seseorang mengembangkan atau menggunakan sebuah algoritme untuk mengimplementasikan programnya, maka ada satu hal yang perlu diperhatikan yaitu kompleksitas. Singkatnya, kompleksitas itu menunjukkan seberapa besar sumber daya yang akan digunakan (dibutuhkan) untuk menjalankan algoritme tersebut sebagai sebuah fungsi dari jumlah data yang diproses. Sebagai contoh, sebuah aplikasi mungkin berjalan dengan baik ketika jumlah datanya hanya 100, tetapi kemudian menjadi sangat lambat atau bahkan tidak jalan ketika jumlah datanya 1000 atau lebih. Nah jumlah data tersebut (100 atau 1000) merupakan sebuah paramter yang menjadi fungsi dari sumber daya yang digunakan. Biasanya ini kita sebut n. Kompleksitas kemudian menjukkan seberapa besar sumber daya sebagai fungsi dari n itu.

Ada algoritme yang lama komputasinya bergantung liner terhadap jumlah data (n) itu. Jadi misalnya untuk 100 data dibutuhkan 100 detik, maka untuk 1000 data dibutuhkan waktu 1000 detik. Bagaimana jika jumlah datanya 10000? Maka dapat diperkirakan waktu yang dibutuhkan adalah 10000 detik. Kompleksitas yang sepert ini disebuh $O(n)$.

Ada juga algoritme yang lama perhitungannya berbanding kuadratis (pangkat dua) terhadap jumlah datanya. Kompleksitas yang model seperti ini disebut $O(n^2)$

```
n=1000
import time

start_time = time.time()
a = 0
for x in range(n):
    a = a + 0
end_time = time.time()
elapsed = end_time - start_time
print("elapsed %s seconds ..." % elapsed)
```

Contoh kode di atas menunjukkan sebuah *loop* dengan 1000 kali putaran. Jumlah loop ditentukan oleh variabel n, yang pada contoh di atas diisi 1000. Di dalam setiap loop ada perintah *dummy*, yaitu variabel a ditambah dengan bilangan nol. Ini hanya sekedar untuk menunjukkan beban komputasi penjumlahan. Tentu saja di dalam dunia nyata, komputasiny lebih berat.

Jika program tersebut dijalankan maka dia akan menghasilkan waktu eksekusi sebanyak $4,5538 \times 10^{-5}$ detik. Angka tepatnya tentu saja bergantung kepada komputer yang digunakan. Ini hanya seba-

gai contoh saja, yang kebetulan diambil dari laptop saya.

Sekarang mari kita tambahkan loop di dalamnya sehingga kodenya menjadi seperti berikut.

```
n=1000
import time

start_time = time.time()
a = 0
for x in range(n):
    for y in range(n):
        a = a + 0
end_time = time.time()
elapsed = end_time - start_time
print("elapsed %s seconds ..." % elapsed)
```

Pada prinsipnya, kita menambahkan sebuah loop lagi di dalam loop sebelumnya. Jadi di dalam loop yang x, kita menambahkan loop y yang jumlah loop-nya juga sama (n). Jika kode ini dijalankan maka dihasilkan keluaran waktu $0,05148$ detik, atau $5,418 \times 10^{-2}$. Perhatikan bahwa waktu yang dibutuhkan untuk mengeksekusi ini 1000 kali lebih besar (lebih lambat) dari waktu sebelumnya. Kompleksitas dari algoritme ini menjadi $O(n^2)$.

Mari kita tambahkan satu loop lagi di dalam loop y sehingga kodenya menjadi seperti berikut.

```
n=1000
import time

start_time = time.time()
a = 0
for x in range(n):
    for y in range(n):
       for z in range(n):
           a = a + 0
end_time = time.time()
elapsed = end_time - start_time
print("elapsed %s seconds ..." % elapsed)
```

Seperti sebelumnya, kode ini kita jalankan juga. Hasilnya adalah $51,1767$ detik. Atau $5,11767 \times 10$. Perhatikan bahwa ini 1000 kali lebih lambat dari sebelumnya dan 1 juta kali lebih lambat dari loop yang pertama kali (yang hanya ada x). Kompleksitas dari algoritme ini adalah $O(n^3)$.

Anda boleh mencoba menggantikan angka variabel n tadi dengan 10000. Namun perlu diingat bahwa waktu yang dibutuhkan

untuk menjalankan kode tersebut menjadi sangat lama. Jika Anda tertarik, coba jalankan. Berapa detik waktu yang dibutuhkan untuk menjalankan itu?

Apa yang kita bahas ini adalah kompleksitas yang dikaitkan dengan lamanya (waktu) eksekusi. Ada juga kompleksitas yang terkait dengan besarnya memori yang dibutuhkan (untuk memproses data). Ini merupakan bahasan yang lebih kompleks, tetapi prinsipnya sama.

1.6 List

Python memiliki struktur data *list*, yang sama dengan *array* (larik) di bahasa pemrograman lainnya. Untuk membuat *list* kita menggunakan tanda siku kotak. Contoh berikut ini menunjukkan penggunaan struktur data *list*.

```
# contoh list di Python
daftar = [7, 1, 5, 3, 2, 4, 6, 8, 3, 5]
print(daftar)
print("list memiliki", len(daftar), "elemen")

# akses elemen yang pertama (dengan indeks 0)
print("daftar[0] =", daftar[0])

# menambahkan data ke list
daftar = daftar + [7, 9]
print(daftar)
```

Ada banyak hal yang dapat kita lakukan dengan menggunakan struktur data *list* ini. Kita dapat menghapus (*remove*) elemen di daftar tersebut. Kita juga dapat memotong (*cut*) elemen yang berada di tengah-tengah. Selain itu juga kita dapat menghitung berapa kemunculan sebuah nilai tertentu pada *list* tersebut.

1.7 Input

Salah satu cara untuk mendapatkan masukan (input) dari pengguna secara interaktif adalah dengan menggunakan fungsi "input" (untuk Python 3.*) atau "raw_input" (untuk Python 2.7).

```
# ini untuk Python 2.7
# gunakan raw_input
nama = raw_input("Masukkan nama Anda: ")
print "Selamat pagi,", nama
```

Perhatikan bahwa kita menggunakan variabel "nama" untuk

menyimpan masukan dari pengguna. Variabel "nama" tersebut mempunyai tipe *string*. Python mengenali secara otomatis.

Mari kita coba tampilkan huruf-huruf yang ada di dalam variabel "nama" tersebut.

```
# for loop bisa menggunakan elemen dari string
# tidak harus indeks angka
for i in nama:
    print i
```

Kita juga dapat membuat statistik kemunculan huruf dari nama (atau teks) yang dimasukkan oleh pengguna. Statistik ini dapat dimanfaatkan untuk proses enkripsi, misalnya. Gunakan program "input" di atas, dan gabungkan dengan kode berikut ini.

```
# associative array: hitung jumlah huruf dan spasi
huruf = {}  # inisialisasi
for key in nama:
    if key in huruf:
        huruf[key] += 1
    else:
        huruf[key]=1
# tampilkan hasil python 2.7
# sorted() agar key-nya diurutkan
# for python 3.* use this: for key, value in d.items():
for key, value in sorted(huruf.iteritems()):
    print key, value
```

Contoh program di atas menggunakan *associative array* atau dalam Python disebut *dictionary*. Pada prinsipnya ini adalah array tetapi dengan menggunakan *immutable object* seperti *string* sebagai indeks atau kuncinya.

Pada contoh tersebut, spasi (*space*) masih dianggap sebagai huruf. Coba ubah sehingga spasi tidak dimasukkan sebagai indeks.

1.8 Pemrosesan Teks

Salah satu manfaat utama dari bahasa pemrograman seperti Python adalah kemampuannya dalam memproses teks (*text processing*). Bahasa pemrograman lainnya, seperti C, tentu saja dapat digunakan untuk melakukan pemrosesan teks. Namun bahasa C lebih "sulit" digunakan karena ada banyak hal yang harus kita ketahui dari awal.

```
# text processing
# memecah kalimat menjadi kata-kata
kalimat = raw_input("Masukkan kalimat yang cukup panjang.\n")
```

```
# pisahkan menjadi kata
kata = kalimat.split()
for k in kata:
    print k
```

Contoh singkat di atas menunjukkan cara memecahkan kalimat menjadi kata-kata. Sebagai catatan, kalimat yang dimaksudkan diakhiri dengan *return*. Untuk memproses kalimat yang lebih panjang dan memiliki *return* harus dilakukan perbaikan. Coba kembangkan program yang dapat menerima masukan dari sebuah berkas.

Dengan menggunakan ide pada bagian sebelumnya, kita dapat menghitung jumlah kemunculan kata tertentu dalam sebuah kalimat. (Perhatikan bahwa "kata" di sini bersifat *case sensitive*. Agar dia tidak bergantung kepada huruf besar dan kecil, semua huruf harus diubah dahulu ke huruf kecil.)

Program ini juga dapat menjadi basis dari sebuah sistem untuk menganalisis sentimen seseorang di media sosial. Pikirkan algoritmanya untuk melakukan hal tersebut.

1.9 Python3

Bagaimana caranya agar kita dapat menggunakan Python3 sebagai *default* dari Python? Cara yang paling mudah adalah dengan menggunakan fitur alias di shell (jika Anda menggunakan variasi dari UNIX).

```
alias python=python3
```

Jika Anda ingin membuat ini menjadi permanen dan Anda menggunakan *bash* sebagai shell Anda, letakkan alias tersebut pada berkas ".bashrc" pada *home directory* Anda (atau pada berkas ".bash_aliases"). Jika Anda menyimpannya di dalam berkas tersebut, maka perubahan baru akan terjadi jika Anda membuat sesi shell baru atau Anda logout dan login kembali. Jika Anda ingin langsung aktif, bisa juga berkas tersebut di-source.

```
source ~/.bashrc
```

Untuk memasang modul-modul di Python3 dapat dilakukan dengan cara memanggil python3 secara eksplisit. Sebagai contoh, untuk memasang modul "numpy" pada (dengan) python3 adalah sebagai berikut.

```
python3 -m pip install numpy
```

Sebagai catatan ada banyak cara untuk memasang modul Python, tetapi cara di atas yang paling konsisten bagi saya.

2
Koding Tingkat Medium

Pada bagian ini akan dibahas berbagai pemrograman Python yang lebih *advanced*. Sebetulnya yang akan dibahas adalah contoh-contoh kode Python dengan menggunakan berbagai paket yang tersedia.

2.1 Argumen CLI

Seringkali kita harus membuat sebuah program dalam bentuk *command line interface* (CLI) yang kemudian membaca argumen yang diberikan. Misalnya kita ingin program kita memproses sebuah berkas yang namanya kita berikan di *command line*.

```
$ program filename.pdf
```

Apa yang kita berikan kepada program di atas disebut *argument*. Pada contoh di atas, *argument*-nya adalah "filename.pdf". Program kita harus dapat membaca *argument* yang kita berikan kepada program tersebut. Bagaimana caranya? Ada banyak caranya. Salah satunya dicontohkan pada contoh berikut ini.

```
# contoh parsing argumen yang diberikan kepada program
# beri nama skrip ini cli-args.py
# cara menjalankan:
#    python3 cli-args.py opsi1 opsi2 opsi3
# opsi bisa banyak

import sys
# periksa jumlah argumennya

numberargs = len(sys.argv)
# tanpa argumen, hasilnya akan 0 (nol)
# nama skrip adalah sys.argv[0]

# print argumen yang diberikan
```

```
for i in range(numberargs):
   print(i, sys.argv[i])
```

Dari contoh tersebut, Anda dapat mengembangkan hal-hal yang lain. Misalnya Anda ingin memastikan bahwa program Anda mendapatkan *argument* dalam jumlah yang cukup (misalnya harus 3). Jika argumen yang diberikan kurang, maka dia akan memberikan petunjuk cara penggunaan skrip (program) kita dan kemudian keluar (dengan exit). (Catatan: biasanya *exit* memiliki nilai tidak nol kalau ada kesalahan. Kalau keluar normal, angkanya nol.)

```
if (numberargs) < 4:
   print("Usage: " + sys.argv[0] + " data1 data2 data3")
exit(1)
```

2.2 *Numpy*

Numpy adalah paket python untuk berbagai aplikasi *scientific*. Di dalamnya ada *N-dimensional object*, *linear algebra*, *Fourier transform*, dan seterusnya. Sebagai contoh, jika kita ingin membangkitkan bilangan random dengan distribusi tertentu (uniform atau normal), maka kita dapat menggunakan paket Numpy ini. Biasanya paket Numpy ini sudah terpasang ketika kita memasang Python, tetapi jika belum terpasang maka modul Numpy ini dapat kita pasang sendiri.

```
$ sudo pip install numpy
```

Contoh-contoh penggunaan paket Numpy akan digabungkan dengan bagian lain. Berikut ini adalah contoh sederhana penggunaan dari Numpy.

```
>>> import numpy as np
>>> a = np.arange(15).reshape(3, 5)
>>> a
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14]])
```

Berikut adalah contoh operasional matriks dengan menggunakan numpy. Pertama, kita dapat membuat matriks *a* dan *b* kemudian menjumlahkan dan mengalikan matriks tersebut.

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \quad B = \begin{bmatrix} 3 & 3 & 3 \\ 5 & 5 & 5 \\ 5 & 3 & 1 \end{bmatrix} \quad C = A + B \quad D = A \times B$$

```
import numpy as np
a = np.array([[1,2,3],[4,5,6],[7,8,9]])
print(a)
b = np.array([[3,3,3],[5,5,5],[5,3,1]])
print(b)
c = a+b
print(c)
d = np.matmul(a, b)
print(d)
```

Keluaran dari perintah di atas adalah seperti ini. Pertama-tama ditampilkan matriks *a* dan *b*, kemudian operasi penambahan dan perkalian. Perhatikan bahwa operasi perkalian matriks menggunakan fungsi *matmul*.

```
 [1 2 3]
 [4 5 6]
 [7 8 9]]

[[3 3 3]
 [5 5 5]
 [5 3 1]]

[[ 4  5  6]
 [ 9 10 11]
 [12 11 10]]

[[ 28  22  16]
 [ 67  55  43]
 [106  88  70]]
```

Contoh-contoh di atas hanya sekedar ilustrasi bagaimana kita menggunakan modul *Numpy*. Secara umum, jika ada pemrosesan data yang multi dimensional dan seringkali terkait dengan operasi matematika, maka Numpy sangat bermanfaat.

2.3 *Matlplotlib*

Salah satu aplikasi yang cukup sering dibutuhkan ketika kita membuat program untuk keperluan penelitian adalah membuat grafik (plot). Salah satu *library* yang baik untuk digunakan adalah *matplotlib*. Paket ini membutuhkan paket lain, yaitu *python-tk*. Untuk itu python-tk ini harus dipasang dulu. Di bawah ini adalah contoh pemasangan python-tk di sistem Linux (berbasis Debian) dengan menggunakan perintah apt-get.

```
$ sudo apt-get install python-tk
$ sudo pip install matplotlib
```

Berikut ini adalah sebuah contoh penggunaan Matplotlib dan Numpy. Pada contoh ini kita akan membuat kumpulan data yang memiliki karakteristik "sekitar" persamaan $Y = Ax + b$. Untuk itu perlu dihasilkan data yang sudah ditambahi atau dikurangi dengan angka random (yang dibuat dengan menggunakan Numpy). (Kode ini diambil dari buku "Getting Started with Tensorflow"[1].) Hasilnya dapat dilihat pada gambar 2.1

[1] Giancarlo Zaccone. *Getting Started with Tensorflow*. Packt Publishing, 2016

```
import numpy as np
import matplotlib.pyplot as plt

# persamaan y = a*x + b
a = 0.25
b = 0.75
jumlah_titik = 300

# buat dua list yang masih kosong
x_point = []
y_point = []

for i in range(jumlah_titik):
    x = np.random.normal(0.0,0.4)
    y = a*x + b + np.random.normal(0.0,0.1)
    x_point.append([x])
    y_point.append([y])

plt.plot(x_point,y_point,'o',label='Random Data')
plt.legend()
plt.show()
```

Jika kode di atas ingin dijalankan di dalam *Jupyter Notebook,* maka baris pertama perlu ditambahkan ini:

```
%matplotlib notebook
```

Data (x dan y) pada contoh di atas dapat disimpan (diekspor) ke berkas dalam format CSV (*comma separated value*) dengan menggunakan Numpy seperti contoh di bawah ini. Berkas "linear-regression.csv" disimpan pada direktori dimana kode ini dijalankan. Variabel *x_point* dan *y_point* akan dimasukkan ke berkas tersebut dengan format yang didefinisikan dalam *fmt*. Pada contoh di bawah ini format yang akan digunakan adalah *floating point* dengan 5 digit di belakang koma. Variabel tersebut dipisahkan dengan menggunakan koma (,) sebagaimana dijabarkan dalam *delimiter*.

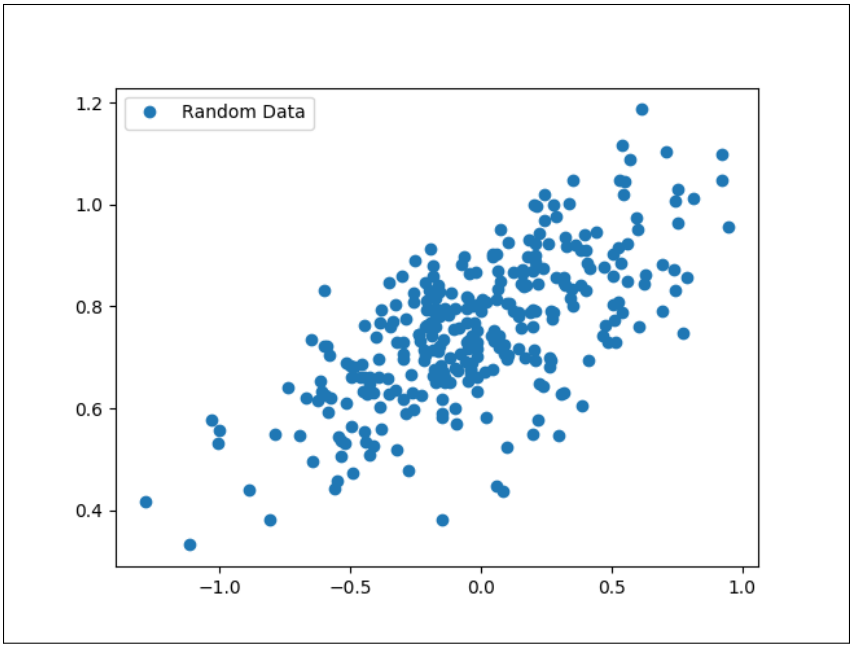

Figure 2.1: Contoh Pembangkitan Random Data

```
np.savetxt("linear-regression.csv", np.column_stack([x_point, y_point]), fmt='%.5f', delimiter=', ')
```

Data di atas dapat dibaca kembali dari berkas CSV dan dilakukan perhitungan (*linear regression*) untuk mencari faktor *gradient* (faktor A) dan b dalam persamaan $Y = a * x + b$. Perhitungan ini membutuhkan modul *Scipy* yang harus dipasang secara terpisah. (Gunakan pip untuk memasang modul scipy itu.)

```
#read CSV of data and calculate a and b
# y = ax + b
import numpy as np
# do not forget to install scipy first: python3 -m pip install scipy
from scipy import stats

my_csv = np.genfromtxt('linear-regression.csv', delimiter=',')
xp, yp = my_csv.transpose()
gradient,intercept,r_value,p_value,std_err=stats.linregress(xp,yp)
print("Gradient and intercept",gradient,intercept)
print("R-squared",r_value**2)
print("p-value",p_value)
```

Jika diperlukan, data tersebut dapat ditampilkan ulang dan garis (lurus) dapat digambarkan pula.

2.4 *Pandas*

Pandas adalah library untuk data processing. Dia banyak digunakan untuk berbagai aplikasi, seperti misalnya di *Artificial Intelligence* (AI) atau *Machine Learning*, statistik, dan operasi yang melibatkan banyak data dalam bentuk tabel. Bayangkan ini seperti menggunakan *spreadsheet* (seperti Excel).

Langkah pertama yang dilakukan adalah memasang Pandas.

```
$ sudo pip install pandas
```

Jika Anda menggunakan Python3, maka gunakan perintah berikut.

```
$ sudo python3 -m pip install pandas
```

Setelah Pandas terpasang, mari kita coba membuat sebuat *Series*. Kali ini dia berisi data bilangan dan *NaN*. Pandas akan secara otomatis membuat indeks dari data tersebut (dengan index bilangan integer)[2].

[2] Contoh lain dapat dilihat di https://pandas.pydata.org/pandas-docs/stable/10min.html

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# create series
ser = pd.Series([1,3,5,7,np.nan,9,11])
print(ser)
```

Hasilnya adalah sebagai berikut. Perhatikan bentuk keluarannya yang seperti tabel dalam *spreadsheet*.

```
0     1.0
1     3.0
2     5.0
3     7.0
4     NaN
5     9.0
6    11.0
dtype: float64
```

2.5 *OpenCV*

OpenCV adalah kumpulan pustaka untuk bidang *computer vision* atau *image processing*. Pustaka ini dapat diakses dengan menggunakan bermacam-macam bahasa pemrograman, termasuk bahasa Python. Untuk memasang pustaka atau modul ini gunakan perintah.

```
pip install opencv-python
```

Berikut ini adalah sebuah contoh penggunaan OpenCV untuk membaca sebuah berkas gambar dan mengubahnya warnanya menjadi abu-abu (greyscale). Perhatikan nama berkas yang dalam contoh ini berada di direktori satu tingkat di atasnya. Ganti nama berkas ini dengan nama yang sesuai dengan berkas yang ingin Anda proses.

```
import cv2
img = cv2.imread('../graphics/br-pixel.png')
grey = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
cv2.imshow('Gambar Asli', img)
cv2.imshow('Gambar grey', grey)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Program akan menampilkan gambar sebelum dan sesudah diproses, kemudian menunggu sampai kita menekan sesuatu di keyboard sebelum akhirnya menutup semua peragaan gambar tersebut. Hasil pemrosesan kode di atas dapat dilihat pada gambar-gambar berikut.

Figure 2.2: Gambar asli

Perhatikan betapa mudahnya menggunakan OpenCV untuk melakukan proses tersebut. Konversi warna itu dapat kita lakukan dengan satu perintah saja. Tanpa pustaka ini kita harus membuat sebuah *loop* yang mengubah setiap piksel dari gambar tersebut.

Modul OpenCV ini dapat digunakan di berbagai aplikasi, termasuk untuk Artificial Intelligence atau Machine Learning.

Figure 2.3: Hasil proses

2.6 *Kriptografi*

Sebagaimana bidang lain, Python memiliki *library* yang lengkap untuk kriptografi. Berikut ini hanya beberapa contoh penggunaan *library* tersebut.

2.6.1 *Fungsi Hash*

Fungsi hash adalah fungsi satu arah yang memberikan tanda (*signature*) dari data digital; *stream of data* dan berkas. Perubahan satu bit saja dari data tersebut akan mengubah nilai dari *hash* yang dihasilkan. Itulah sebabnya fungsi *hash* dapat digunakan untuk menjamin integritas data.

Ada banyak algoritma fungsi hash. Algoritma yang terkenal adalah MD5 dan SHA. Saat ini MD5 sudah dianggap tidak layak lagi karena sudah ditemukan *collision*, yaitu nilai *hash* yang sama untuk data yang berbeda. SHA 256 merupakan algoritma yang dianggap cocok saat ini.

```
unix$ echo "beli 10000" | shasum -a 256
375a6c46228994656932f4aa17d9ae50f21da75a31ff17f8517c255c06cba809 -

unix$ cat pesan1.txt
beli 10000
unix$ shasum -a 256 pesan1.txt
```

```
375a6c46228994656932f4aa17d9ae50f21da75a31ff17f8517c255c06cba809 pesan1.txt

unix$ cat pesan2.txt
beli 1000
unix$ shasum -a 256 pesan2.txt
5901bccc6a0556fac2b4a164ef831a7ed4ceddeb60c6ddde1162f5a40b9d2917 pesan2.txt
```

Contoh kode Python untuk hal di atas adalah sebagai berikut:

```
# Contoh fungsi hash
import hashlib
h = hashlib.sha256("beli 10000\n")
print h.hexdigest()
```

Salah satu pemanfaatan "baru" dari fungsi *hash* ini adalah pada algoritma *Blockchain* yang digunakan pada *Bitcoin*. Sedikit cerita tentang hal ini ada di blog saya [3].

[3] https://rahard.wordpress.com/2018/03/10/berburu-hash/

3
Triks

Ada beberapa hal menarik dari Python. Pada bagian ini akan ditampilkan berbagai trik tersebut.

3.1 Server Web

Python dapat kita gunakan sebagai server web sederhana yang dapat memerikan list direktori. Fitur ini sering saya gunakan untuk transfer berkas antar mesin. Misalnya saya ingin mengambil sebuah berkas dari satu komputer ke komputer yang lain. Maka komputer dimana berkas tersebut berada saya jalankan perintah ini (pada direktori dimana berkas tersebut berada).

```
python -m http.server 8008
```

Perintah di atas akan membuat sebuah server web yang berjalan pada *port* 8008. Anda dapat menggunakan port yang berbeda. Akan ditampilkan daftar berkas yang ada di direktori tersebut. Saya tinggal memilih berkas yang dimaksudkan dan *Save as ....*

Fitur server web ini juga sering saya gunakan untuk proses *debugging* sebuah aplikasi yang menggunakan protokol web. Untuk melihat data yang dikirimkan oleh aplikasi tersebut, saya arahkan dia ke web server ini dan kemudian saya perhatikan apa yang ditampilkan (yang diminta oleh aplikasi tersebut).

3.2 Membaca Berkas Konfugurasi config.ini

Seringkali ada kebutuhan untuk menyimpan *credential* (userid, password) dalam aplikasi. Sebagai contoh, kita ingin membuat koneksi ke server lain (server database, server FTP, dan sejenisnya) maka dibutuhkan informasi tentang nama atau nomor IP dari server yang dituju, userid dan password, dan informasi lainnya lagi. Seringkali informasi ini ditanam di dalam kode (source code).

Apa beberapa masalah terkait dengan pendekatan ini (menanam credential) di dalam kode sumber. Pertama, jika terjadi perubahan (misal ada pergantian nomor IP dari server yang dituju) maka kita harus mencari di dalam kode mana saja terdapat penanaman informasi tersebut dan tentu saja harus diperbaharui. Bayangkan kalau informasi ini terdapat di beberapa berkas. Bagaimana kita tahu itu ada dimana? Jika konfigurasi ini hanya ada di satu berkas, misalnya di berkas "config.ini", maka operator hanya perlu melakukan perubahaan di berkas ini.

Kedua, perubahan terhadap kode sangat berisiko. Beberapa kali terjadi kasus seseorang mengubah kode dari sebuah aplikasi (dengan bayangan minor update), ternyata aplikasi menjadi tidak berjalan. Ini dapat terjadi ketika update kekurangan tanda petik, misalnya, sehingga aplikasi malah menjadi "error".

Jika kode ingin dibagikan, misal menggunakan github, maka credential sudah terpisahkan dengan kode sehingga kemungkinan tersebar juga lebih kecil. Tentu saja masih mungkin terjadi jika berkas config.ini berada dalam direktori yang dibagikan (share) dan tidak dimasukkan ke dalam berkas (.gitignore) dalam kasus github.

Berikut ini adalah contoh kode yang lazim ditemui, yaitu menanamkan *credential* di dalam kode sumber. Perhatikan bahwa alamat server, userid, dan password tertanam dalam kode.

```
from ftplib import FTP

SERVER = '192.168.4.29'
NAMA = 'jabar'
GEMBOK = 'juara2020'

ftp = FTP(SERVER)
ftp.login(user=NAMA, passwd=GEMBOK)
ftp.dir()
ftp.quit()
```

Cara yang lebih baik adalah dengan menyimpan credential tersebut ke dalam sebuah berkas konfigurasi. Berikut ini adalah sebuah contoh isi berkas "config.ini".

```
[aplikasiku]
   server = 192.168.4.29
   user = jabar
   password = juara2020
[queue]
   server = rabbitmq-server
   user = rabbitku
   password = sangatrahasia6677
```

Berkas ini dapat dibaca dari aplikasi dengan cara berikut, dengan asumsi bahwa berkas "config.ini" berada dalam direktori yang sama dengan aplikasi ini. Untuk lebih meningkatkan keamanan, seringkali letak berkas konfigurasi berada di luar struktur direktori ini (di atasnya). Jadi variabel CONF dapat diisi dengan "../config.in".

```
from ftplib import FTP
from configparser import ConfigParser

CONF='config.ini'
config = ConfigParser()
# return filename
config.read(CONF)

SERVER = config.get('aplikasiku','server')
NAMA = config.get('aplikasiku','user')
GEMBOK = config.get('aplikasiku','password')

ftp = FTP(SERVER)
ftp.login(user=NAMA, passwd=GEMBOK)
ftp.dir()
ftp.quit()
```

Penjelasan mengenai ini ada dalam video berikut.

https://www.youtube.com/watch?v=jze3DC6ohRc

3.3 Menulis berkas dalam format Unicode

Unicode merupakan alternatif pengkodean karakter untuk karakter yang tidak terdapat di kode ASCII. Bagaimana membuat berkas yang berisi karaktor Unicode? Berikut ini adalah salah satu contohnya.

```
unicode_text = u'BUDI Rahardjo'
myencoded = unicode_text.encode('utf-16-le')
#myencoded = unicode_text.encode('utf-16-be')

a_file = open("budi-unicode.txt", "wb")
a_file.write(myencoded)

a_file = open("budi-unicode.txt", "r", encoding='utf-16-le')
#a_file = open("budi-unicode.txt", "r", encoding='utf-16-be')
# reads contents of a file
contents = a_file.read()
print(contents)
```

Kode di atas menyimpan string 'Budi Rahardjo' ke dalam sebuah berkas dalam format UTF-16 dengan enkoding *little endian* (le) atau *big endian* (be)[1]. Untuk melihat bagaimana data tersebut disimpan dalam berkas, dapat digunakan peritah "hexdump" atau "xxd". Perhatikan bahwa jika Anda menggunakan LE atau BE posisi bytes akan tertukar. Silahkan dicoba.

[1] Mengenai little endian atau big endian akan dibahas secara terpisah.

```
hexdump budi-unicode.txt

0000000 0042 0055 0044 0049 0020 0052 0061 0068
0000010 0061 0072 0064 006a 006f
000001a
```

4
Bibliography

[1] Giancarlo Zaccone. *Getting Started with Tensorflow*. Packt Publishing, 2016.