

# Dasar Pemrograman Python pada Keilmuan Data

DR. ARYA ADHYAKSA WASKITA

# Daftar Isi

Daftar Isi	i
Daftar Gambar	iii
Daftar Tabel	v
Daftar Program	v
KATA PENGANTAR	vii
<b>1 Pendahuluan</b>	<b>1</b>
1.1 Sejarah singkat . . . . .	1
1.2 Kenapa Python? . . . . .	1
1.3 Keilmuan data . . . . .	2
<b>2 Instalasi Python</b>	<b>5</b>
2.1 Interpreter Python . . . . .	5
2.2 Anaconda . . . . .	9
<b>3 <i>Hello World!</i></b>	<b>17</b>
3.1 Pendahuluan . . . . .	17
3.2 Variabel dan jenis datanya . . . . .	17
3.3 Kondisi dan Perulangan . . . . .	18
<b>4 Dasar Pemrograman Python</b>	<b>21</b>
4.1 Pendahuluan . . . . .	21
4.2 Struktur Data . . . . .	23
4.2.1 List . . . . .	24
4.2.2 Tuple . . . . .	25
4.2.3 Dictionary . . . . .	26
4.3 Operator . . . . .	27
4.3.1 Aritmatika . . . . .	27
4.3.2 Penugasan . . . . .	28
4.3.3 Perbandingan . . . . .	29

4.3.4	Logika . . . . .	29
4.3.5	Identitas . . . . .	29
4.3.6	Keanggotaan . . . . .	30
<b>5</b>	<b>Mendefinisikan fungsi dan menangani kesalahan</b>	<b>33</b>
5.1	Mendefinisikan fungsi . . . . .	33
5.1.1	Fungsi Umum . . . . .	33
5.1.2	Lambda . . . . .	33
5.2	Menangani kesalahan ( <i>exception</i> ) . . . . .	34
<b>6</b>	<b>Membuat dan mengelola modul</b>	<b>35</b>
6.1	Pendahuluan . . . . .	35
6.2	Membuat modul . . . . .	36
6.3	Jejak pencarian . . . . .	39
<b>7</b>	<b>Interaksi berkas</b>	<b>41</b>
7.1	Pendahuluan . . . . .	41
7.2	Berkas tunggal . . . . .	41
7.3	Berkas jamak . . . . .	44
7.4	Latihan . . . . .	45
7.4.1	Berkas tunggal . . . . .	45
7.4.2	Berkas jamak . . . . .	45
<b>8</b>	<b>Interaksi Basisdata</b>	<b>47</b>
8.1	MySQL . . . . .	47
8.2	PostgreSQL . . . . .	47
8.3	MongoDB . . . . .	47
	<b>Bibliografi</b>	<b>49</b>

# Daftar Gambar

1.1	Guido van Rossum . . . . .	1
1.2	<i>Word cloud</i> untuk saling keterkaitan dalam keilmuan data [Braschler et al., 2019] . . . . .	2
1.3	Hirarki kebutuhan kecerdasan buatan . . . . .	3
2.1	Dialog instalasi <i>interpreter</i> Python . . . . .	5
2.2	Pilihan paket pendukung sebelum instalasi dilakukan . . . . .	6
2.3	Dialog selama proses instalasi berlangsung . . . . .	6
2.4	Dialog tanda selesai instalasi . . . . .	6
2.5	Lokasi instalasi <i>interpreter</i> Python . . . . .	7
2.6	<i>Interpreter</i> Python siap digunakan . . . . .	7
2.7	Daftar paket yang terpasang . . . . .	8
2.8	Hasil <b>upgrade</b> pip . . . . .	8
2.9	Daftar terakhir paket terpasang . . . . .	8
2.10	Daftar menu aplikasi pendukung Python . . . . .	9
2.11	Aplikasi <b>IDLE</b> . . . . .	9
2.12	Pilihan <i>platform</i> instalasi Anaconda . . . . .	10
2.13	Dialog pembuka instalasi . . . . .	10
2.14	Menyetujui kesepakatan . . . . .	11
2.15	Pilihan pengguna Anaconda . . . . .	11
2.16	Target instalasi . . . . .	11
2.17	Menjadikan Anaconda sebagai sistem utama Python . . . . .	12
2.18	Proses instalasi . . . . .	12
2.19	Instalasi selesai . . . . .	12
2.20	Anaconda yang tampil di Menu Ms. Windows© . . . . .	13
2.21	Aplikasi <b>Jupyter</b> . . . . .	13
2.22	Terminal pada aplikasi <b>Jupyter</b> . . . . .	14
2.23	Python <b>Shell</b> pada aplikasi <b>Jupyter</b> . . . . .	14
2.24	Aplikasi <b>Spyder</b> . . . . .	15
3.1	Menampilkan teks <i>Hello World!</i> pada Python <b>shell</b> . . . . .	17
3.2	Hasil menjalankan Program 3.5 . . . . .	19

4.1	Python <code>shell</code> sedang menerima perintah . . . . .	22
4.2	Variabel <code>a</code> sebagai obyek . . . . .	22
4.3	Menampilkan dokumentasi obyek <code>integer a</code> . . . . .	23
4.4	Menampilkan dokumentasi obyek <code>integer a</code> menggunakan fungsi <code>help</code> . . . . .	23
4.5	Proses penambahan elemen <code>list</code> . . . . .	24
4.6	Perbandingan penambahan elemen <code>list</code> menggunakan fungsi (a). <code>append</code> dan (b). <code>extend</code> . . . . .	25
4.7	Penambahan karakter 'x' ke variabel <code>a</code> di posisi pertama . . . . .	25
4.8	Mengeluarkan elemen tertentu dari <code>list</code> . . . . .	25
4.9	Mengeluarkan elemen terakhir dari variabel <code>list</code> . . . . .	25
4.10	Beberapa operasi yang dilakukan pada variabel <code>tuple</code> . . . . .	26
4.11	Menambahkan elemen ke variabel <code>dictionary</code> . . . . .	26
4.12	Mengeluarkan pasangan <i>key-value</i> dari variabel <code>dictionary</code> . . . . .	27
4.13	Operasi aritmatika . . . . .	27
4.14	Operator pembagian pada variabel berjenis <code>int</code> pada Python <code>3.x</code> . . . . .	28
4.15	Operator pembagian pada variabel berjenis <code>int</code> pada Python <code>2.x</code> . . . . .	28
4.16	Perbedaan respon evaluasi dua variabel berbeda dengan nilai yang sama . . . . .	30
4.17	Perbedaan respon evaluasi dua variabel identik dengan nilai yang sama . . . . .	30
5.1	Contoh sederhana fungsi <code>lambda</code> . . . . .	34
6.1	Fungsi faktorial dipanggil dua kali . . . . .	38
7.1	Informasi statistik dataset iris . . . . .	44

# Daftar Tabel

4.1	Operator aritmatika di Python . . . . .	27
4.2	Operator penugasan di Python . . . . .	28
4.3	Operator perbandingan di Python . . . . .	29
4.4	Operator logika di Python . . . . .	29
4.5	Operator identitas di Python . . . . .	30
4.6	Operator keanggotaan di Python . . . . .	31



# Daftar Program

3.1	Menampilkan <i>Hello World!</i> dengan C . . . . .	17
3.2	Kondisi dengan pasangan <b>if-else</b> . . . . .	18
3.3	Kondisi dengan pasangan <b>if-elif-else</b> . . . . .	18
3.4	Menampilkan angka 1 s/d 9 dengan <b>for</b> . . . . .	18
3.5	Menampilkan teks tertentu sebanyak jumlah elemen dalam <b>list</b> . . . . .	18
3.6	<b>for</b> bersarang . . . . .	19
3.7	Pengulangan <i>indefinite</i> . . . . .	19
4.1	Mendefinisikan fungsi sederhana . . . . .	31
5.1	Mendefinisikan fungsi sederhana . . . . .	33
5.2	Menghentikan perulangan ketika didapati operasi yang tidak valid . . . . .	34
6.1	Menghitung nilai faktorial . . . . .	35
6.2	Menghitung nilai faktorial yang diterapkan sebagai fungsi . . . . .	36
6.3	Fungsi faktorial sebagai modul . . . . .	36
6.4	<i>Script</i> yang menggunakan modul <b>faktorial</b> . . . . .	37
6.5	Modul <b>faktorial</b> lengkap dengan fungsi ujinya . . . . .	37
6.6	<i>Script</i> yang menggunakan modul <b>faktorial</b> yang di dalamnya ada fungsi uji . . . . .	38
6.7	Modul <b>faktorial</b> lengkap dengan fungsi ujinya dan fungsi <b>main</b> . . . . .	38
7.1	Membaca dataset iris dan menampilkan isinya dengan format tertentu . . . . .	42
7.2	Menghitung total data untuk setiap kelas . . . . .	44





# Kata Pengantar

Dengan berkembang pesatnya keilmuan data (*data science*), mahasiswa dan dosen perlu menguasai *tools* yang dapat menunjang aktifitas mereka untuk mengeksplorasi keilmuan tersebut. Salah satu *tools* yang umum digunakan dalam keilmuan data berbasis pemrograman python. Karena alasan tersebut, buku elektronik ini disusun.

Secara umum, diktat ini dibagi ke dalam bagian pendahuluan yang membahas tentang sejarah singkat Python yang dilanjutkan ke bagian instalasi. Instalasi ini, meskipun sangat sederhana, terutama pada sistem operasi Linux, dapat menjadi sangat merepotkan bagi beberapa mahasiswa, terutama ketika mereka menggunakan sistem operasi Windows. Karena itu, instalasi akan dilakukan di sistem operasi Windows. Bagian selanjutnya adalah dasar-dasar pemrograman Python, terutama struktur data (`list`, `tuple` dan `dictionary`), interaksi dengan *file* dan basis data, hingga membuat modul yang dapat digunakan kembali. Akhirnya, selamat mencoba pengalaman baru.

Serpong, 28 Agustus 2020

Dr. Arya Adhyaksa Waskita

# Bab 1

## Pendahuluan

### 1.1 Sejarah singkat

Python dibangun oleh Guido van Rossum (Gambar 1.1<sup>1</sup>) pada sekitar tahun 1980 di *Centrum Wiskunde & Informatica* (CWI) di Belanda [Hunt, 2019]. Nama Python diambil dari program TV favorit Guido yang berjudul ”Monty Pythons Flying Circus” yang tayang pada kisaran tahun 1969-1974.



**Gambar 1.1:** Guido van Rossum

### 1.2 Kenapa Python?

Berikut adalah beberapa jawaban dari pertanyaan tersebut.

- *Multiplatform*. Python adalah bahasa pemrograman yang tersedia pada sejumlah *platform* sistem operasi seperti GNU Linux, Windows dan Mac. Selain itu, Python juga tersedia pada *platform* perangkat bergerak seperti Android dan *embedded system*<sup>2</sup>.
- Mudah. Python merupakan bahasa pemrograman yang mudah karena **syntax** yang sederhana, sehingga mudah dikuasai bahkan oleh pengguna yang tidak memiliki latar belakang pendidikan formal di bidang ilmu komputer.

---

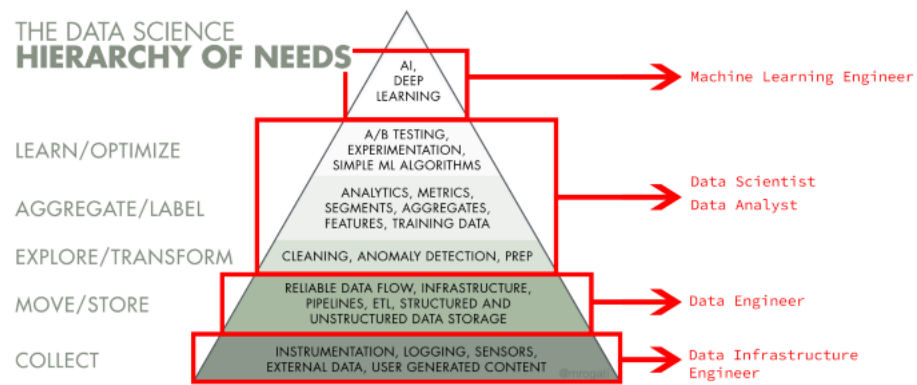
<sup>1</sup><https://gvanrossum.github.io/images/guido-headshot-2019.jpg>

<sup>2</sup><https://wiki.python.org/moin/EmbeddedPython>

- ### 1.3 Keilmuan data

Algoritma dan metode yang diperlukan untuk menghasilkan sistem pendukung keputusan saat ini banyak memanfaatkan pembelajaran mesin dan kecerdasan buatan, selain statistik sebagai pondasi keilmuannya. Gambar 1.3 menunjukkan piramida di mana ilmu data berperan<sup>5</sup> dalam hirarki yang pertama kali dibuat oleh Monica Rogati<sup>6</sup>. Berdasarkan hirarki tersebut, pelatihan ini mengasumsikan bahwa data telah tersedia (terutama dalam format CSV). Selanjutnya, peserta harus mampu berinteraksi dengan data tersebut, terutama untuk tujuan praproses sebelum nantinya diolah dengan algoritma tertentu memanfaatkan pustaka berbasis Python.

<sup>6</sup><https://medium.com/hackernoon/the-ai-hierarchy-of-needs-18f111fcc007>



**Gambar 1.3:** Hirarki kebutuhan kecerdasan buatan



## Bab 2

# Instalasi Python

### 2.1 Interpreter Python

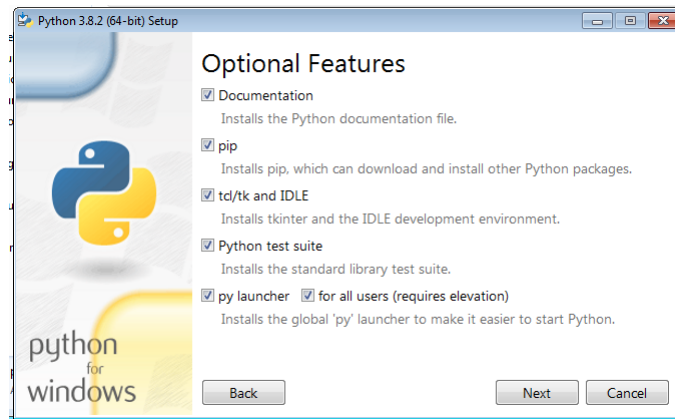
Seperti telah dijelaskan di bagian Pengantar, instalasi *interpreter* Python dilakukan di sistem operasi Windows 7. Tahapan instalasi ini mengasumsikan bahwa tidak ada kendala apapun terkait sistem operasi. Selanjutnya mahasiswa diminta untuk mengunduh *interpreter* Python melalui laman <https://www.python.org/downloads/> sesuai kebutuhannya.

Mengeksekusi unduhan tersebut akan memunculkan dialog seperti pada Gambar 2.1. Pastikan untuk memilih konfigurasi **PATH** secara otomatis agar ketika proses instalasi selesai, *interpreter* Python dapat dijalankan dari mana saja di sistem komputer masing-masing. Untuk kondisi di mana terjadi kesalahan, akan muncul dialog yang memberi kita kesempatan untuk melihat *log*. Buka log tersebut dan lihat sumber dari kesalahan instalasi yang sedang terjadi.

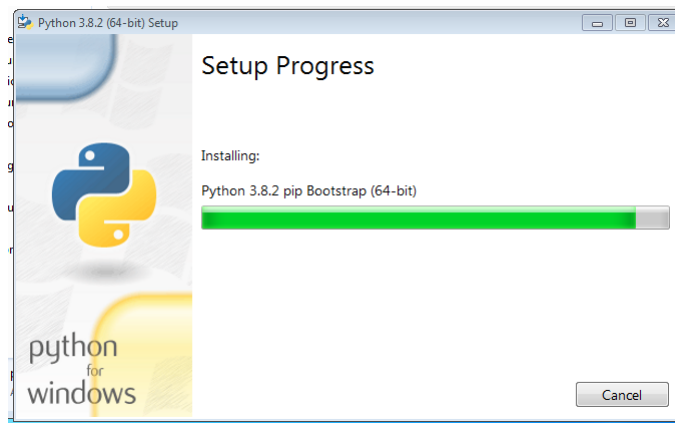


**Gambar 2.1:** Dialog instalasi *interpreter* Python

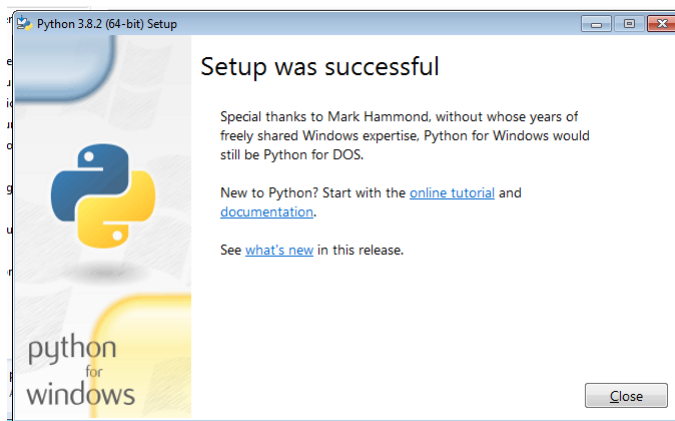
Pilihan opsi *Customize installation* akan menampilkan dialog seperti Gambar 2.2. Pastikan semua pilihan dipilih. Kemudian, selama proses instalasi berlangsung, pengguna akan disuguhkan dialog seperti Gambar 2.3. Tunggu sampai dialog tanda selesai dikeluarkan seperti pada Gambar 2.4.



Gambar 2.2: Pilihan paket pendukung sebelum instalasi dilakukan



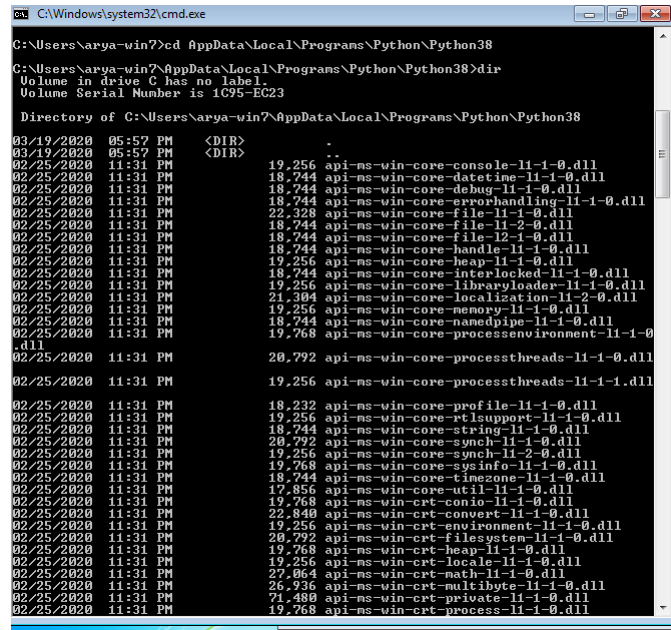
Gambar 2.3: Dialos selama proses instalasi berlangsung



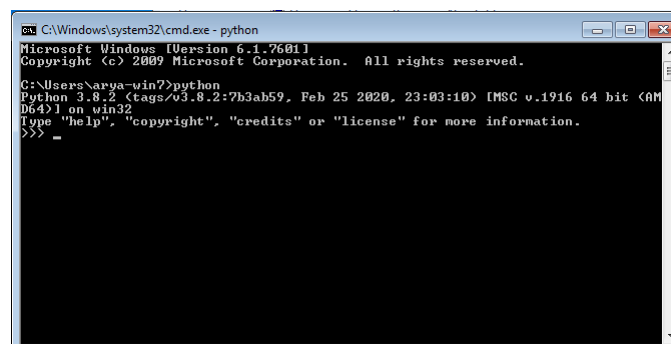
Gambar 2.4: Dialog tanda selesai instalasi



Seperti telah ditunjukkan pada Gambar 2.1 tentang informasi lokasi *interpreter* Python di letakkan, dapat juga dibuktikan melalui aplikasi CMD seperti Gambar 2.5. Sedangkan *interpreter* Python dapat diujicobakan dengan menuliskan perintah `python` di aplikasi CMD. Akan muncul dialog seperti Gambar 2.6. *Interpreter* Python siap digunakan, ditandai dengan munculnya karakter `>>>`.

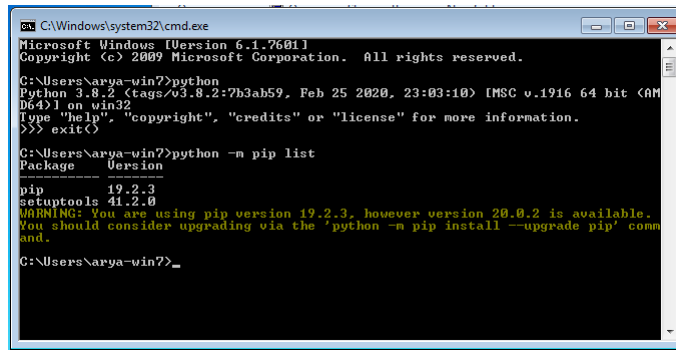


Gambar 2.5: Lokasi instalasi *interpreter* Python



Gambar 2.6: *Interpreter* Python siap digunakan

Tahapan selanjutnya adalah instalasi pustaka `scikit-image`. Proses instalasinya dilakukan dengan aplikasi pengelola paket Python yang bernama `pip`. Silakan lihat Gambar 2.2. `pip` ada di urutan kedua dari fitur tambahan. `pip` dapat digunakan untuk melihat paket apa saja yang telah terpasang di sistem kita. Caranya dengan menjalankan perintah `python -m pip list` seperti ditunjukkan Gambar 2.7.



```

C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\arya-win7>python
Python 3.8.2 (tags/v3.8.2:7b3ab59, Feb 25 2020, 23:03:10) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> exit()

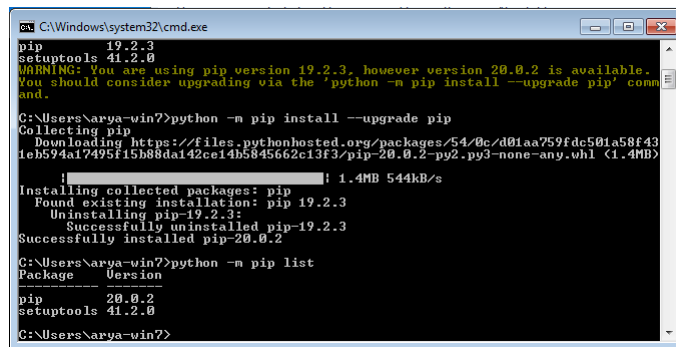
C:\Users\arya-win7>python -m pip list
Package            Version
-----
pip                19.2.3
setuptools         41.2.0
WARNING: You are using pip version 19.2.3, however version 20.0.2 is available.
You should consider upgrading via the 'python -m pip install --upgrade pip' command.

C:\Users\arya-win7>_

```

Gambar 2.7: Daftar paket yang terpasang

pip dapat juga digunakan untuk meng-upgrade paket yang telah terpasang, bahkan dirinya sendiri. Untuk meng-upgrade paket pip itu sendiri, dapat dilakukan dengan menjalankan perintah `python -m pip install --upgrade pip` seperti Gambar 2.8. Perhatikan versi pip yang ada di Gambar 2.7 dan Gambar 2.8.



```

C:\Windows\system32\cmd.exe

C:\Users\arya-win7>python -m pip install --upgrade pip
Collecting pip
  Downloading https://files.pythonhosted.org/packages/54/0c/d01aa759fd501a58f431eb594a17495f15b88da142ce14b5845662c13f3/pip-20.0.2-py2.py3-none-any.whl (1.4MB)
    |#####| 1.4MB 544kB/s
Installing collected packages: pip
Found existing installation: pip 19.2.3
Uninstalling pip-19.2.3:
  Successfully uninstalled pip-19.2.3
Successfully installed pip-20.0.2

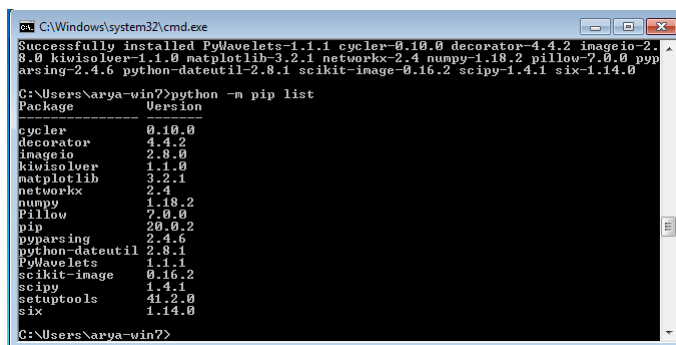
C:\Users\arya-win7>python -m pip list
Package            Version
-----
pip                20.0.2
setuptools         41.2.0

C:\Users\arya-win7>

```

Gambar 2.8: Hasil upgrade pip

Setelah selesai, kita dapat kembali melihat daftar paket yang terpasang melalui pengelolaan pip yang ditunjukkan Gambar 2.9.



```

C:\Windows\system32\cmd.exe

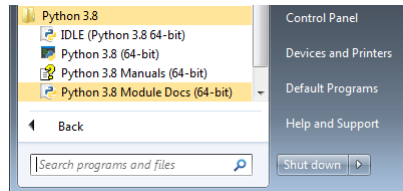
C:\Users\arya-win7>python -m pip list
Package            Version
-----
cycler              0.10.0
decorator           4.4.2
imageio             2.8.0
kivisolver          1.1.0
matplotlib          3.2.1
networkx            2.4
numpy               1.18.2
Pillow              7.0.0
pip                20.0.2
pyparsing           2.4.6
python-dateutil     2.8.1
PyWavelets          1.1.1
scikit-image        0.16.2
scipy               1.4.1
setuptools          41.2.0
six                 1.14.0

C:\Users\arya-win7>

```

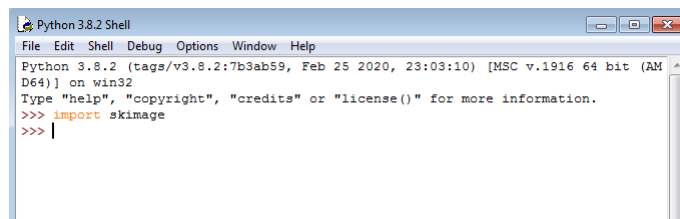
Gambar 2.9: Daftar terakhir paket terpasang

Menu aplikasi pendukung Python akan muncul seperti Gambar 2.10. Menu kedua pada Gambar 2.10 akan memunculkan aplikasi CMD yang sama dengan yang ditunjukkan Gambar 2.6, tetapi tanpa perlu memanggil perintah `python` terlebih dahulu. CMD secara otomatis akan memunculkan Python `shell` seperti Gambar 2.6.



Gambar 2.10: Daftar menu aplikasi pendukung Python

IDLE adalah antarmuka *interpreter* Python seperti ditunjukkan Gambar 2.11. Dalam Gambar 2.11 juga terlihat bahwa kita berhasil meng-*import* pustaka `scikit-image`, yang dalam IDLE di Windows 7 disebut sebagai `skimage`. Jika Anda sedang menggunakan Ubuntu, kemudian menggunakan pustaka `scikit-image` yang diperoleh dari *repository* Ubuntu (bukan dari `pip`), pustaka `scikit-image` juga di-*import* dengan nama `skimage`. Berhasilnya sebuah pustaka Python di-*import* adalah ketika tidak ada komentar yang muncul setelah perintah `import` tersebut.

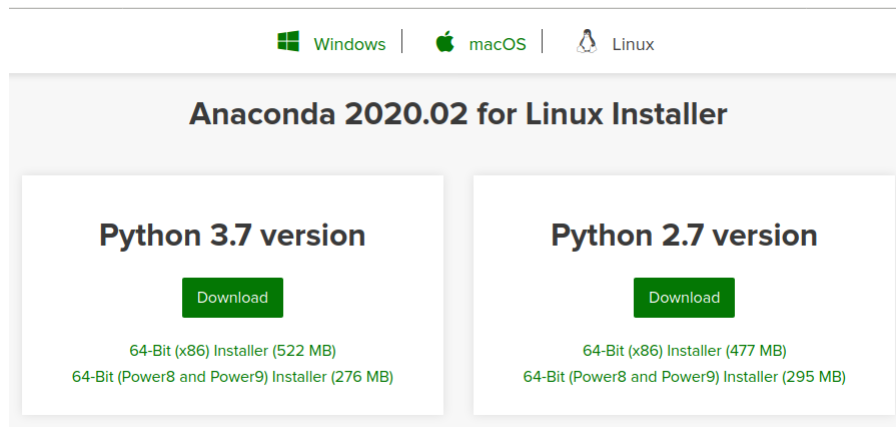


Gambar 2.11: Aplikasi IDLE

Selanjutnya, jika ditemukan petunjuk untuk masuk ke Python `Shell`, Anda dapat menggunakan aplikasi IDLE, atau menggunakan terminal (di Linux)/CMD (di Windows) dengan terlebih dahulu menjalankan perintah `python`.

## 2.2 Anaconda

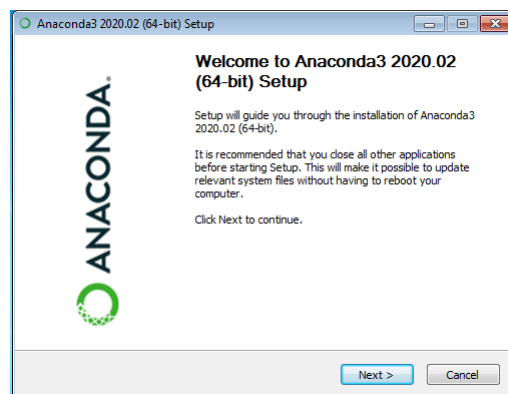
Selain pilihan manual seperti yang telah dijelaskan di Sub bab 2.1, Anaconda bisa menjadi opsi lain yang lebih bersifat otomatis. Saya menyebutnya otomatis karena Anaconda sejumlah pustaka Python, terutama yang banyak digunakan di *Data Mining*, *Machine Learning* atau *Data Science* telah dikemas di dalam Anaconda. Bahkan beberapa editor yang populer untuk Python juga dikemasnya. Anaconda bahkan mengemasnya khusus untuk *platform* yang berbeda. Anda dapat menghubungi alamat <https://www.anaconda.com/> untuk mengunduh aplikasinya. Sesuaikan kebutuhan Anda dengan pilihan yang ada seperti ditunjukkan Gambar 2.12.



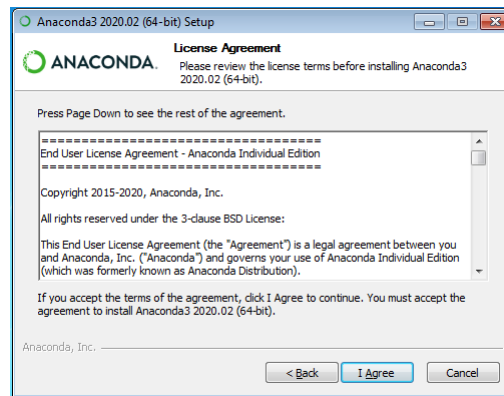
**Gambar 2.12:** Pilihan *platform* instalasi Anaconda

Instalasi Anaconda akan menghadirkan dialog seperti ditunjukkan Gambar 2.13 - Gambar 2.19. Anaconda akan meletakkan pustaka di lokasi `C:\\ProgramData\\Anaconda3` yang berbeda dengan `pip` seperti terlihat di Gambar 2.16. Sedangkan di Gambar 2.18 terlihat sejumlah pustaka penting seperti `scikit-image` dan `scikit-learn` tengah diinstal.

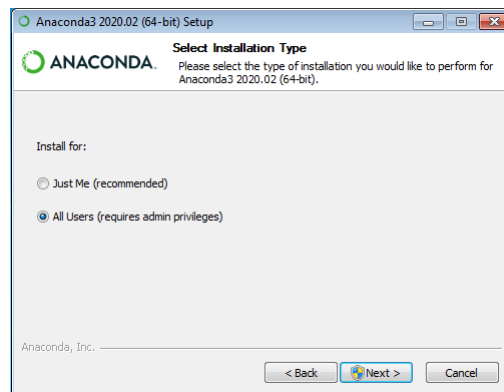
aplikasi ini akan menghadirkan antarmuka seperti tampak



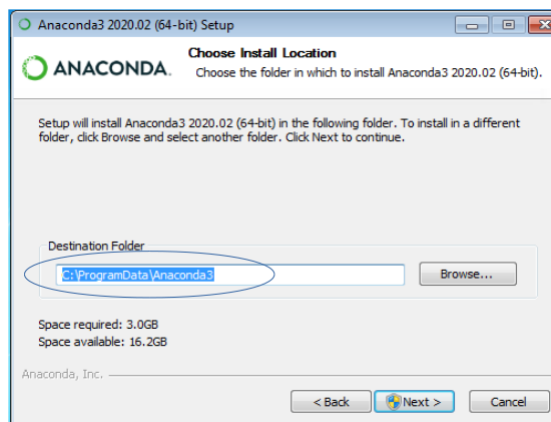
**Gambar 2.13:** Dialog pembuka instalasi



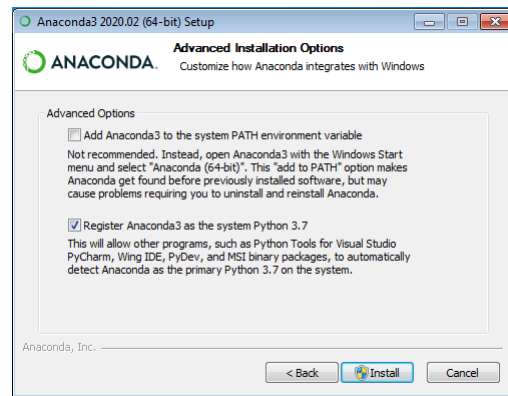
Gambar 2.14: Menyetujui kesepakatan



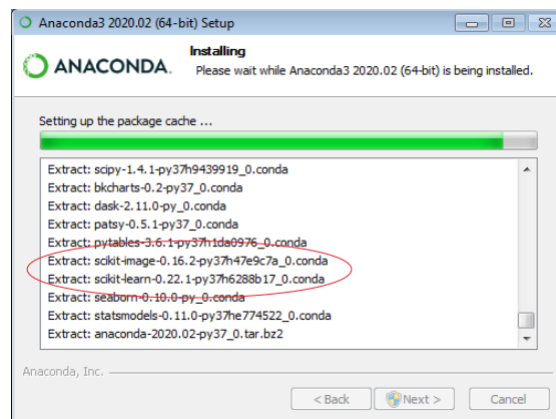
Gambar 2.15: Pilihan pengguna Anaconda



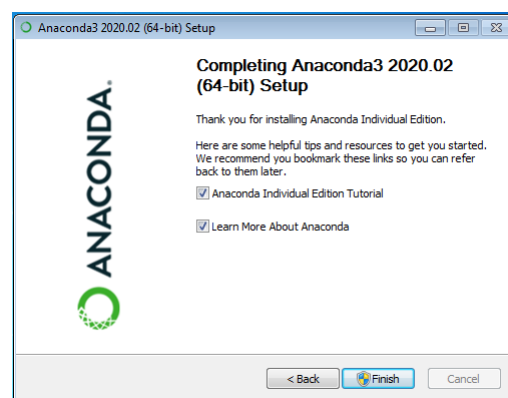
Gambar 2.16: Target instalasi



Gambar 2.17: Menjadikan Anaconda sebagai sistem utama Python



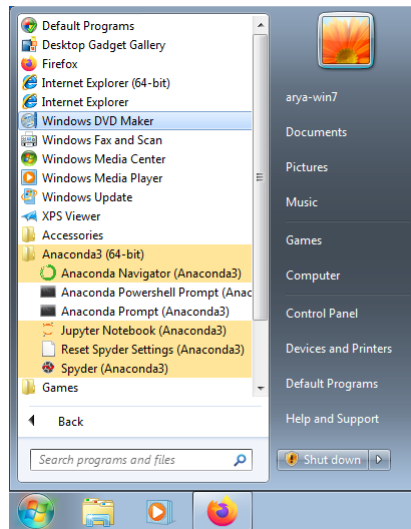
Gambar 2.18: Proses instalasi



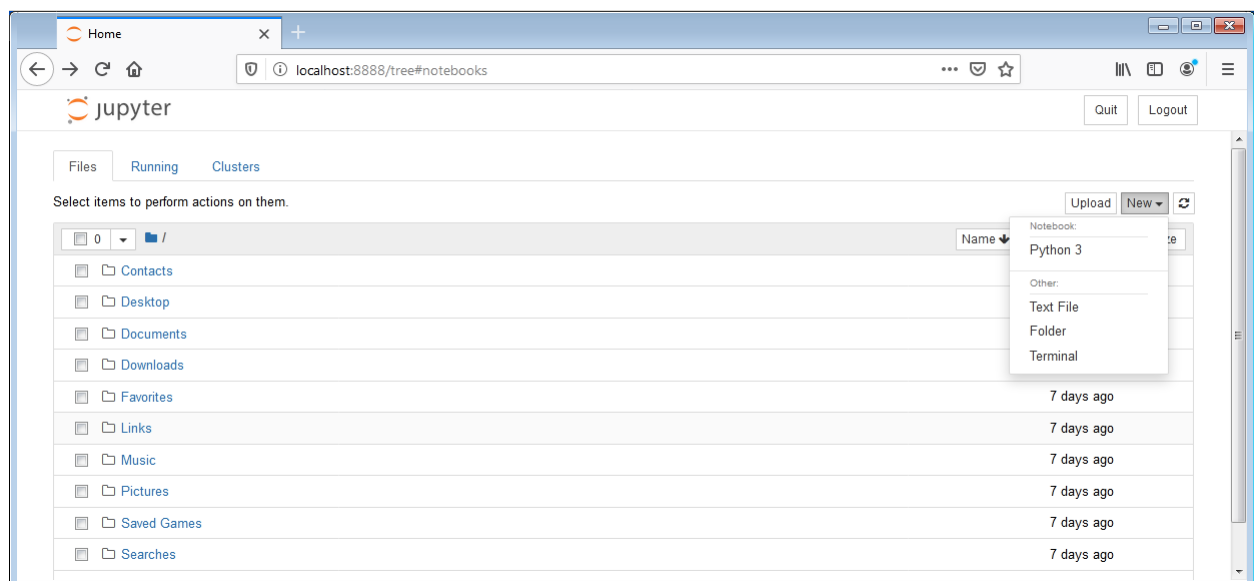
Gambar 2.19: Instalasi selesai

Instalasi Anaconda akan membuat menu seperti pada Gambar 2.20. Di situ terlihat sejumlah aplikasi yang dapat digunakan untuk mengembangkan kode komputer berbasis Python seperti

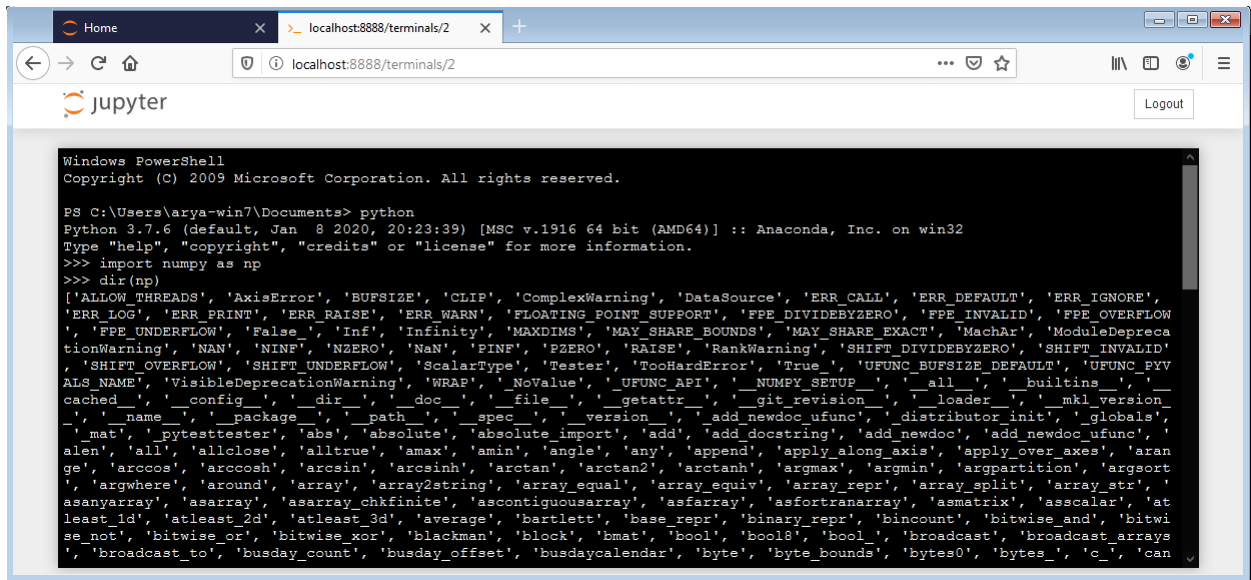
Jupyter dan Spyder. Untuk Jupyter, aplikasi ini akan menghadirkan antarmuka seperti tampak pada Gambar 2.21. Di sisi kanan atas terlihat beberapa opsi antarmuka untuk mengelola proyek Python dengan Jupyter, seperti Terminal Gambar 2.22 atau Python Shell di bawah Jupyter seperti Gambar 2.23 yang perannya seperti IDLE di Gambar 2.11. Sedangkan untuk Spyder, akan tampak antarmuka seperti Gambar 2.24.



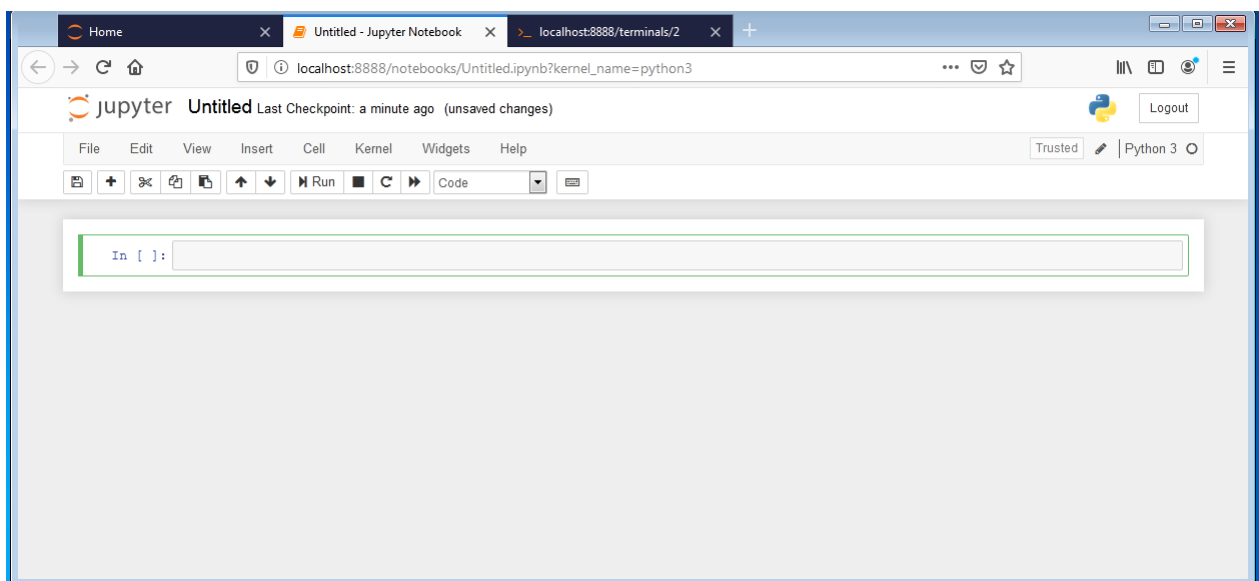
Gambar 2.20: Anaconda yang tampil di Menu Ms. Windows©



Gambar 2.21: Aplikasi Jupyter

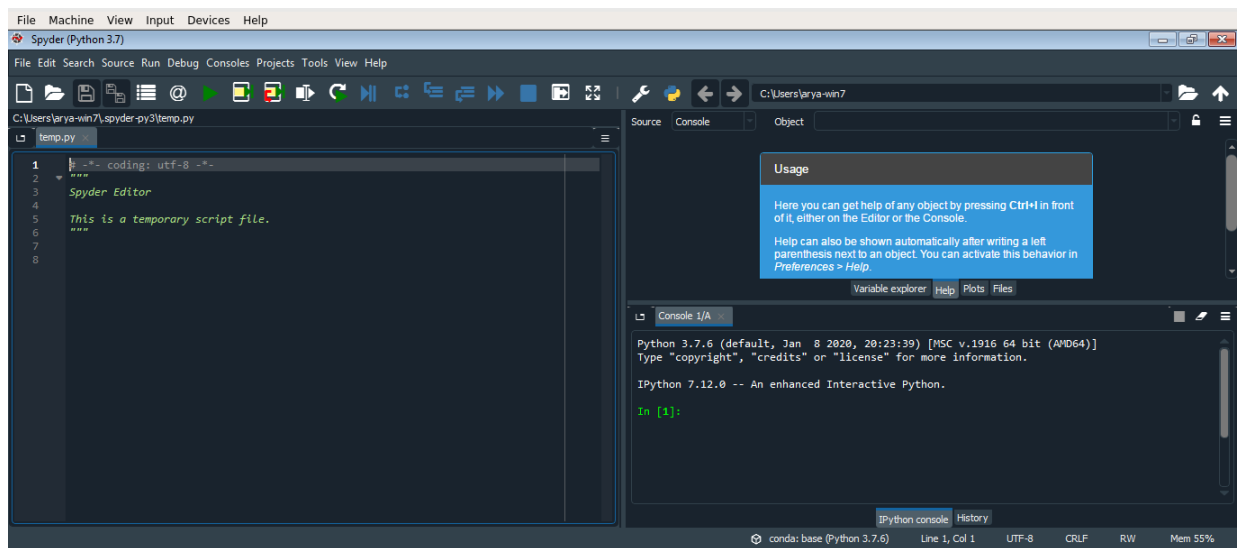


Gambar 2.22: Terminal pada aplikasi Jupyter



Gambar 2.23: Python Shell pada aplikasi Jupyter





Gambar 2.24: Aplikasi Spyder



## Bab 3

# *Hello World!*

### 3.1 Pendahuluan

Untuk mulai memprogram dalam Python, buka *interpreter* Python hingga muncul dialog seperti Gambar 2.6. Setelah itu, ketikkan perintah `print('Hello World!')` yang hasilnya ditunjukkan pada Gambar 3.1. Jika dibandingkan dengan Program 3.1, maka Python paling efisien dalam penggunaan perintah. Hasilnyapun seketika diperoleh tanpa perlu melakukan kompilasi.

```
arya@arya-pc:~$ python
Python 3.8.2 (default, Jul 16 2020, 14:00:26)
[GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> print('Hello World!')
Hello World!
>>> █
```

**Gambar 3.1:** Menampilkan teks *Hello World!* pada Python shell

**Program 3.1:** Menampilkan *Hello World!* dengan C

```
1 #include<stdio.h>
2 int main(){
3     printf("Hello Wordl!\n");
4     return 0;
5 }
```

### 3.2 Variabel dan jenis datanya

Variabel python dapat dideklarasikan ketika akan digunakan (*on demand*). Selain itu, kita tidak perlu mendeskripsikan tipe data dari variabel tersebut. Sebagai contoh, deklarasi variabel `a=2` sudah cukup untuk mendeklarasi variabel `a` sebagai variabel yang berjenis `int`. Untuk mengetahui jenis variabel, gunakan perintah `type`, contohnya `type(a)`.

### 3.3 Kondisi dan Perulangan

Kondisi adalah syarat yang harus dipenuhi untuk dijalankannya perintah tertentu. Kita dapat menggunakan pasangan `if-else` untuk menerapkan kondisi tertentu. Program 3.2 contoh penerapan dua kondisi. Sedangkan Program 3.3 menunjukkan penerapan lebih dari dua kondisi dengan `if-elif-else`.

**Program 3.2:** Kondisi dengan pasangan `if-else`

```
1 a='Arya Adhyaksa Waskita'
2 if len(a)>=19:
3     print('Ya, dia orangnya!')
4 else:
5     print('Bukan!')
```

**Program 3.3:** Kondisi dengan pasangan `if-elif-else`

```
1 import random
2 a=random.randint(0,30)
3 if a<=10:
4     print('antara 0-10')
5 elif a>10 and a<=20:
6     print('antara 11-20')
7 else:
8     print('antara 20-30')
```

Perulangan dapat dilakukan dengan jumlah tertentu (*definite*), bisa juga tidak (*indefinite*). Perhatikan Program 3.4. Jika perulangan dilakukan `in range(10)`, maka angka yang akan ditampilkan dimulai dari 0. Sedangkan batas atasnya adalah sebanyak nilai argumen `in range` dikurangi 1. Perulangan juga dapat dilakukan berdasarkan jumlah elemen `list` seperti ditunjukkan Program 3.5. Sementara eksekusinya menghasilkan Gambar 3.2. Keduanya adalah contoh dari perulangan yang bersifat *definite*.

**Program 3.4:** Menampilkan angka 1 s/d 9 dengan `for`

```
1 for i in range(1,10):
2     print(i)
```

**Program 3.5:** Menampilkan teks tertentu sebanyak jumlah elemen dalam `list`

```
1 a=[1, 3, 5, 7, 9]
2 for i in a:
3     print('Hello World!')
```

```
Hello World!  
Hello World!  
Hello World!  
Hello World!  
Hello World!  
arya@arya-pc:
```

**Gambar 3.2:** Hasil menjalankan Program 3.5

Perhatikan Program 3.6 untuk mencetak kalimat sebanyak nilai elemen `list`.

**Program 3.6:** `for` bersarang

```
1 a=[1, 3, 5, 7, 9]  
2 for i in a:  
3     print(i)  
4     for j in range(i):  
5         print('Hello World!')
```

Untuk perulangan *indefinite*, Program 3.7 adalah contohnya. Dalam Program 3.7, perulangan dilakukan hingga nilai random yang dihasilkan sama dengan 0. Proses ini bermanfaat ketika kita perlu melakukan operasi pembagian, di mana penyebutnya tidak boleh sama dengan 0. Dengan pendekatan lain seperti di sub bab 5.2, deteksi kesalahan dapat dilakukan.

**Program 3.7:** Pengulangan *indefinite*

```
1 import random  
2 i=0  
3 a=random.randint(0,10)  
4 while(a!=0):  
5     a=random.randint(0,10)  
6     i=i+1  
7 print('Perulangan dilakukan sebanyak '+str(i)+' kali')
```



## Bab 4

# Dasar Pemrograman Python

### 4.1 Pendahuluan

Bahasa pemrograman Python memiliki 4 sifat dasar berikut<sup>1</sup>.

1. *Interpreter*. Python diproses oleh *interpreter*, sehingga tidak perlu dikompilasi untuk menjalankannya. Hal ini seperti dijumpai pada bahasa pemrograman PHP yang sangat populer itu.
2. Interaktif. Anda dapat berinteraksi dengan Python dengan memberikannya perintah satu per satu melalui Python `shell`. Setiap perintah yang diberikan langsung akan direspon. Selain itu, Python bersifat *self explained*. Jika ada fungsi dari suatu obyek yang tidak kita ketahui, kita bisa mempelajarinya langsung dari dokumentasi di Python `shell`.
3. Berorientasi obyek. Ada semacam slogan bahwa *''Everything is object in Python''*. Seperti telah dipahami melalui kuliah Rekayasa Perangkat Lunak, orientasi obyek menyebabkan variabel dan fungsi (sering disebut sebagai *state* dan *behavior*) terkemas dalam sebuah obyek, sehingga memudahkan pengelolaan variabel. Fungsi yang melekat pada sebuah obyek juga dapat diturunkan dari satu obyek ke obyek lain sehingga tidak perlu dideklarasikan ulang. Namun, fitur orientasi obyek ini pemberlakuannya bagi pemrogram tidak seketat seperti yang dilakukan di **Java**. Jika **Java** mengharuskan pemrogram mendeklarasikan kelas untuk membuat program yang bahkan sangat sederhana, maka Python tidak mengharuskannya.
4. Bahasa pemrograman untuk pemula. Hal ini disebabkan karena Python sangat sederhana, tidak memerlukan banyak deklarasi yang seringkali menyulitkan, bahkan menakutkan bagi pemula. Selain itu, Python juga mendukung pengembangan aplikasi untuk banyak *platform*, dari aplikasi *embedded* hingga *web* dan *mobile*.

Untuk sifat dasar pertama dan kedua, dapat dilihat ilustrasinya di Gambar 4.1. Dalam Gambar 4.1, Python `shell` dipanggil dengan perintah `python3`. Hal tersebut disebabkan karena

---

<sup>1</sup><https://www.tutorialspoint.com/python/index.htm>

Ubuntu (yang sedang digunakan adalah Ubuntu 18.04) secara *default* menyertakan Python versi 2.x. Sedangkan untuk Python versi 3.x harus dijalankan dengan perintah `python3`. Di Gambar 4.1 terlihat bahwa ada dua perintah yang diberikan secara berurutan. Tetapi, Python akan meresponnya satu per satu. Sedangkan untuk keluar dari Python `shell`, berikan perintah `exit()`.

```
arya@arya-pc:~$ python3
Python 3.6.9 (default, Nov 7 2019, 10:44:02)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> print('Hello world!')
Hello world!
>>> 3+7
10
>>> exit()
_
```

**Gambar 4.1:** Python `shell` sedang menerima perintah

Untuk sifat dasar ketiga dapat diilustrasikan melalui Gambar 4.2. Kita dapat mengetahui jenis obyek dari variabel `a` dengan fungsi `type(a)`. Sedangkan untuk melihat fungsi dan variabel apa saja yang terkandung pada variabel `a`, kita dapat menggunakan fungsi `dir(a)`. Tetapi, meskipun semuanya di dalam Python adalah obyek, penggunaan Python tidak mengharuskan kita mendeklarasi kelas secara eksplisit. Dengan menuliskan perintah `a=3`, Python tahu bahwa obyek `a` adalah obyek dari kelas `integer`. Bahkan, di Gambar 4.1, operasi aritmatika dapat dilakukan tanpa mendeklarasi variabel.

```
arya@arya-pc:~$ python3
Python 3.6.9 (default, Nov 7 2019, 10:44:02)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> a=3
>>> dir(a)
['_abs_', '_add_', '_and_', '_bool_', '_ceil_', '_class_', '_delatt',
'_dir_', '_divmod_', '_doc_', '_eq_', '_float_', '_floor_', '_f',
'_floordiv_', '_format_', '_ge_', '_getattr_', '_getnewargs_', '_g',
'_hash_', '_index_', '_init_', '_init_subclass_', '_int_', '_in',
'_vert_', '_le_', '_lshift_', '_lt_', '_mod_', '_mul_', '_ne_', '_ne',
'_g_', '_new_', '_or_', '_pos_', '_pow_', '_radd_', '_rand_', '_rdi',
'_mod_', '_reduce_', '_reduce_ex_', '_repr_', '_rfloordiv_', '_rlshift',
'_rmod_', '_rmul_', '_ror_', '_round_', '_rpow_', '_rrshift_', '_r',
'_rshift_', '_rsub_', '_rtruediv_', '_rxor_', '_setattr_', '_sizeof_',
'_str_', '_sub_', '_subclasshook_', '_truediv_', '_trunc_', '_xor_',
'_bit_length_', '_conjugate_', '_denominator_', '_from_bytes_', '_imag_', '_numerator_',
'_real_', '_to_bytes_']
>>> type(a)
<class 'int'>
>>>
```

**Gambar 4.2:** Variabel `a` sebagai obyek

Di Gambar 4.2 terlihat ada entitas yang diawali dan/atau diakhir dengan karakter dua *underscore* ('`__`') atau sering disebut sebagai *dunder*<sup>2</sup> (*double underscore*) oleh komunitas pemrogram Python. Hal tersebut merupakan bagian dari PEP (*Python Enhancement Proposals*) ke-8 tentang *Style Guide for Python Code*<sup>3</sup>.

Di Gambar 4.2 juga terlihat bahwa obyek `a` memiliki fungsi `__doc__`. Fungsi inilah yang akan memberikan penjelasan singkat kepada kita tentang obyek yang sedang menjadi perhatian.

<sup>2</sup><https://dbader.org/blog/meaning-of-underscores-in-python>

<sup>3</sup><https://www.python.org/dev/peps/pep-0008/>



Untuk menggunakannya, jalankan perintah `a.__doc__` seperti ditunjukkan Gambar 4.3. Dengan `a` adalah nama variabel untuk obyek yang sedang menjadi perhatian.

```
arya@arya-pc:~$ python3
Python 3.6.9 (default, Nov  7 2019, 10:44:02)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> a=3
>>> a.__doc__
"int(x=0) -> integer\nint(x, base=10) -> integer\n\nConvert a number or string to
an integer, or return 0 if no arguments\nare given. If x is a number, return
x.__int__(). For floating point\nnumbers, this truncates towards zero.\n\nIf x
is not a number or if base is given, then x must be a string,\nbytes, or bytearray
instance representing an integer literal in the\ngiven base. The literal can
be preceded by '+' or '-' and be surrounded\nby whitespace. The base defaults
to 10. Valid bases are 0 and 2-36.\nBase 0 means to interpret the base from the
string as an integer literal.\n\n>>> int('0b100', base=0)\n4"
>>>
```

Gambar 4.3: Menampilkan dokumentasi obyek `integer` `a`

Format dokumentasi seperti yang ditunjukkan pada Gambar 4.3 sulit untuk dipahami. Pendekatan lain untuk mempelajari dokumentasi sebuah pustaka adalah dengan menggunakan fungsi `help`. Untuk kasus seperti Gambar 4.3, perintah yang dijalankan adalah `help(a)` (**BUKAN** `a.__doc__`). Hasilnya ditunjukkan pada Gambar 4.4. Untuk keluar dari modus dokumentasi tersebut, pengguna tinggal memberi perintah `q` setelah tanda titik dua (Gambar 4.4). Sedangkan untuk melihat isi dokumentasi selanjutnya pengguna dapat menggunakan tombol spasi di papan ketik.

```
Help on int object:

class int(object)
| int(x=0) -> integer
| int(x, base=10) -> integer
|
| Convert a number or string to an integer, or return 0 if no arguments
| are given. If x is a number, return x.__int__(). For floating point
| numbers, this truncates towards zero.
|
| If x is not a number or if base is given, then x must be a string,
| bytes, or bytearray instance representing an integer literal in the
| given base. The literal can be preceded by '+' or '-' and be surrounded
| by whitespace. The base defaults to 10. Valid bases are 0 and 2-36.
| Base 0 means to interpret the base from the string as an integer literal.
| >>> int('0b100', base=0)
| 4
|
| Methods defined here:
|
| __abs__(self, /)
|     abs(self)
|
| __add__(self, value, /)
|     Return self+value.
|
| __and__(self, value, /)
|     Return self&value.
|
| __bool__(self, /)
|     self != 0
|
| :
```

Gambar 4.4: Menampilkan dokumentasi obyek `integer` `a` menggunakan fungsi `help`

## 4.2 Struktur Data

Struktur data yang dimaksud di sini adalah data *array*/larik dan sejenisnya, serta cara penggunaannya. Tidak jarang, fungsi dalam pustaka *scikit-image* menerima argumen atau mengemba-

likan nilai dalam bentuk data *array* atau sejenisnya.

### 4.2.1 List

*List* adalah *array* yang paling banyak digunakan. Kita dapat menyimpan sejumlah nilai, dari tipe apapun ke dalam *list*, bahkan menambah atau mengurangi isinya. Untuk yang pernah mempelajari bahasa pemrograman C, tentu paham betapa sulitnya melakukan hal tersebut di C. Untuk C++ *list* dapat diterapkan lebih mudah dengan bantuan *standard template library*<sup>4</sup>

Sebuah variabel *list*, misalnya *a*, diinisiasi dengan perintah *a=[]*. Maka, variabel *a* memiliki sejumlah fungsi yang bisa dilihat dengan perintah *dir(a)*. Diktat ini hanya akan membahas fungsi-fungsi yang sering digunakan saja. Fungsi lain bisa dipelajari sendiri dengan bantuan perintah *help(a.nama\_fungsi)*, dengan *a* adalah obyek *list*.

1. **append**. Fungsi ini menambahkan elemen baru ke variabel *list*. Perhatikan Gambar 4.5. Variabel *a* yang awalnya kosong, kemudian diisi satu per satu menggunakan perintah **append**. Variabel *a* terakhir memiliki dua elemen, masing-masing bertipe *integer* dan *character*.

```
>>> a=[]
>>> a.append(3)
>>> a
[3]
>>> a.append('3')
>>> a
[3, '3']
>>> █
```

Gambar 4.5: Proses penambahan elemen *list*

2. **extend**. Fungsi ini memiliki tugas yang sama dengan **append** dengan sedikit perbedaan. Perhatikan Gambar 4.6. Di Gambar 4.6(a), variabel *a* ditambahkan sebuah elemen berupa variabel *list b* menggunakan fungsi **append**. Variabel *b* yang telah memiliki dua elemen ditambahkan ke variabel *a* sebagai satu elemen. Hal tersebut terlihat dari dijalankannya perintah *len(a)*.

Sementara di Gambar 4.6(b), proses yang sama dilakukan menggunakan fungsi **extend**. Fungsi **extend** akan menambahkan variabel *b* ke dalam variabel *a* tidak sebagai *list* secara keseluruhan, tetapi menambahkan masing-masing elemen variabel *b* ke dalam *a*. Itu sebabnya, hasil penambahan *b* ke dalam *a* membuat *a* saat ini memiliki dua elemen.

3. **insert**. Selain menambahkan elemen ke variabel *list* di posisi akhir, penambahan elemen juga dapat dilakukan di posisi tertentu. Perhatikan Gambar 4.7. Penambahan karakter 'x' pada posisi pertama dari *list* dilakukan dengan perintah *a.insert(0,'x')*. Hal ini disebabkan karena indeks dari elemen *list* dimulai dari 0.

<sup>4</sup>[https://en.wikipedia.org/wiki/Standard\\_Template\\_Library](https://en.wikipedia.org/wiki/Standard_Template_Library)

<pre>&gt;&gt;&gt; a=[] &gt;&gt;&gt; b=[1,2] &gt;&gt;&gt; a.append(b) &gt;&gt;&gt; a [[1, 2]] &gt;&gt;&gt; len(a) 1</pre>	<pre>&gt;&gt;&gt; a=[] &gt;&gt;&gt; b=[1,2] &gt;&gt;&gt; a.extend(b) &gt;&gt;&gt; a [1, 2] &gt;&gt;&gt; len(a) 2</pre>
(a)	(b)

**Gambar 4.6:** Perbandingan penambahan elemen `list` menggunakan fungsi (a). `append` dan (b). `extend`

```
>>> a
[1, 2]
>>> a.insert(0,'x')
>>> a
['x', 1, 2]
>>>
```

**Gambar 4.7:** Penambahan karakter 'x' ke variabel `a` di posisi pertama

4. `remove`. Selain menambahkan elemen ke variabel `list`, kita dapat juga membuang salah satu elemen yang ada di posisi tertentu di dalam `list`. Perhatikan figurename 4.8. Perintah `remove` digunakan untuk mengeluarkan elemen tertentu dari `list`. Jika ada lebih dari satu elemen yang sama yang akan dikeluarkan, maka elemen terpilih untuk dikeluarkan adalah elemen yang muncul pertama kali pada `list`.

```
>>> a=['b','c','b','a']
>>> a.remove('b')
>>> a
['c', 'b', 'a']
>>> a.remove('b')
>>> a
['c', 'a']
>>>
```

**Gambar 4.8:** Mengeluarkan elemen tertentu dari `list`

5. `pop`. Fungsi ini akan mengeluarkan elemen terakhir dari `list`. Perhatikan Gambar 4.9.

```
>>> a=[1,2,3,4]
>>> a.pop()
4
>>> a
[1, 2, 3]
>>>
```

**Gambar 4.9:** Mengeluarkan elemen terakhir dari variabel `list`

### 4.2.2 Tuple

`Tuple` adalah jenis *array* selain `list` yang di Python dideklarasikan dengan perintah `a=()`. Operasi pada `tuple` lebih cepat dilakukan jika dibandingkan dengan `list`. Hal ini disebabkan

karena `tuple` bersifat statis karena elemen yang ada di dalamnya tidak dapat diubah, kecuali yang bersifat *mutable*. Karena bersifat statis, deklarasi variabel `a=()` tidak akan bermanfaat. Perhatikan Gambar 4.10.

Di Gambar 4.10, sebuah variabel `a` memiliki empat elemen, di mana elemen ke-4 merupakan sebuah `list`. Elemen ke-4 diakses dengan indeks 3 (karena indeks `tuple` dimulai dari 0). Ketika diakses, isi dari elemen ke-4 tersebut dapat diubah karena bersifat *mutable*. Sebaliknya, ketika elemen lain (dalam hal ini elemen ke-2) akan diubah nilainya, Python menolaknya.

```
>>> a
(1, 2, 3, ['x', 'b'])
>>> a[3]
['x', 'b']
>>> a[3][1]='y'
>>> a
(1, 2, 3, ['x', 'y'])
>>> a[1]
2
>>> a[2]=7
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
>>>
```

**Gambar 4.10:** Beberapa operasi yang dilakukan pada variabel `tuple`

### 4.2.3 Dictionary

`Dictionary` merupakan *array* yang elemen penyusunnya merupakan pasangan *key-value*. Setiap elemen akan diindeks berdasarkan *key*. `Dictionary` dideklarasikan menggunakan perintah `a={}`. Untuk menambah elemen ke variabel `dictionary`, gunakan perintah seperti Gambar 4.11.

```
>>> a={}
>>> a['nama']='Arya Adhyaksa Waskita'
>>> a
{'nama': 'Arya Adhyaksa Waskita'}
>>>
```

**Gambar 4.11:** Menambahkan elemen ke variabel `dictionary`

Kita juga dapat mengeluarkan sebuah elemen dari variabel `dictionary`. Karena elemennya merupakan pasangan *key-value*, maka ketika dikeluarkan, pasangan *key-value* tersebut tidak ada lagi di variabel `dictionary`. Perhatikan Gambar 4.12, elemen yang memiliki *key* berupa karakter `'nama'` akan dikeluarkan menggunakan fungsi `pop`. Karena memerlukan argumen berupa *key*, maka fungsi `pop` dapat mengeluarkan elemen yang posisinya di mana saja di dalam variabel `dictionary`, tidak harus di posisi terakhir. Program 7.1 menunjukkan contoh penggunaan `dictionary` yang lebih kompleks.

```

>>> a['npm']='40081'
>>> a
{'nama': 'Arya Adhyaksa Waskita', 'npm': '40081'}
>>> a.pop('nama')
'Arya Adhyaksa Waskita'
>>> a
{'npm': '40081'}
>>>

```

**Gambar 4.12:** Mengeluarkan pasangan *key-value* dari variabel *dictionary*

## 4.3 Operator

### 4.3.1 Aritmatika

Tabel 4.1 berisi operator aritmatika yang dimiliki Python<sup>5</sup>. Untuk menggunakannya, variabel *a* dan *b* pada Tabel 4.1 harus berjenis yang sama. Perhatikan Gambar 4.13. Tentunya, operasi seperti itu hanya berlaku untuk operator penjumlahan karena karakter memang tidak dapat menerima operator aritmatika.

**Tabel 4.1:** Operator aritmatika di Python

Operator	Nama	Contoh
+	Penjumlahan	a+b
-	Pengurangan	a-b
*	Perkalian	a*b
/	Pembagian	a/b
%	Modulus	a%b
**	Ekspensial	a**b
//	Pembagian dengan pembulatan ke bawah	a//b

```

>>> a='c'
>>> b='5'
>>> a+b
'c5'
>>>

```

**Gambar 4.13:** Operasi aritmatika

Gambar 4.14 menunjukkan contoh operasi aritmatika pembagian. Karena tidak didefinisikan secara eksplisit melalui jenis variabel, maka nilai sebuah variabel akan menjadi penentu jenisnya.

<sup>5</sup>[https://www.w3schools.com/python/python\\_operators.asp](https://www.w3schools.com/python/python_operators.asp)

Tetapi, meski variabel `a` dan `b` berjenis `int`, hasil pembagian tetap berjenis `float`. Ini adalah fitur dari Python 3.x. Di Python 2.x, salah satu variabel harus diberi nilai dari jenis `float` untuk menghasilkan nilai yang juga berjenis `float`. Perhatikan Gambar 4.15.

```
>>> a=5
>>> b=3
>>> c=a/b
>>> c
1.6666666666666667
>>> type(c)
<class 'float'>
>>>
```

**Gambar 4.14:** Operator pembagian pada variabel berjenis `int` pada Python 3.x

```
>>> a=5
>>> b=3
>>> a/b
1
>>> b=3.0
>>> a/b
1.6666666666666667
>>>
```

**Gambar 4.15:** Operator pembagian pada variabel berjenis `int` pada Python 2.x

### 4.3.2 Penugasan

Tabel 4.2 menunjukkan operator penugasan (*assignment*) yang dimiliki Python.

**Tabel 4.2:** Operator penugasan di Python

Operator	Contoh	Bentuk lain
<code>=</code>	<code>x=7.5</code>	<code>x=7.5</code>
<code>+=</code>	<code>x+=2</code>	<code>x=x+2</code>
<code>-=</code>	<code>x-=2</code>	<code>x=x-2</code>
<code>*=</code>	<code>x*=3.1</code>	<code>x=x*3.1</code>
<code>/=</code>	<code>x/=5</code>	<code>x=x/5</code>
<code>%=</code>	<code>x%=2</code>	<code>x=x%2</code>
<code>//=</code>	<code>x//=2</code>	<code>x=x//2</code>
<code>**=</code>	<code>x**=3</code>	<code>x=x**3</code>

### 4.3.3 Perbandingan

Operator perbandingan digunakan untuk menyatakan terjadinya kondisi tertentu. Perhatikan lagi sub bab 3.3. Tabel 4.3 menunjukkan operator perbandingan yang dimiliki Python.

**Tabel 4.3:** Operator perbandingan di Python

Operator	Fungsi	Contoh
<code>==</code>	Kesamaan	<code>a==b</code>
<code>!=</code>	Ketidaksamaan	<code>a!=b</code>
<code>&lt;</code>	Lebih kecil dari	<code>a &lt; b</code>
<code>&gt;</code>	Lebih besar dari	<code>a &gt; b</code>
<code>&lt;=</code>	Lebih kecil dari atau sama dengan	<code>a &lt;=</code>
<code>&gt;=</code>	Lebih besar dari atau sama dengan	<code>a &gt;=</code>

### 4.3.4 Logika

Tabel 4.4 menunjukkan operator logika yang dimiliki Python.

**Tabel 4.4:** Operator logika di Python

Operator	Deskripsi	Contoh
<code>and</code>	<b>True</b> jika kedua pernyataan <b>True</b>	<code>a==2 and b==3</code>
<code>or</code>	<b>True</b> jika salah satu pernyataan <b>True</b>	<code>a==2 or b==3</code>
<code>not</code>	Negasi	<code>not(a==2 and b==3)</code> (tergantung nilai logika awal)

### 4.3.5 Identitas

Tabel 4.5 menunjukkan operator identitas yang dimiliki Python.

Perhatikan Gambar 4.16. Meski variabel `a` dan `b` bernilai sama (tentunya juga berjenis sama), tetapi mereka bukanlah obyek yang sama. Sehingga evaluasi menggunakan operator `is` menghasilkan nilai berbeda. Bandingkan dengan apa yang ditunjukkan Gambar 4.17. Karena variabel `b` mengacu pada variabel `a`, maka evaluasi identitas `a` dan `b` menghasilkan nilai yang sama.

**Tabel 4.5:** Operator identitas di Python

Operator	Fungsi	Contoh
<code>is</code>	True jika kedua variabel merujuk pada obyek yang sama	<code>a is b</code>
<code>is not</code>	True jika kedua variabel merujuk pada obyek yang berbeda	<code>a is not b</code>

```

>>> a='Arya Adhyaksa Waskita'
>>> b='Arya Adhyaksa Waskita'
>>>
>>> if a==b:
...     print('Ya')
...
Ya
>>> if a is b:
...     print('Ya')
... else:
...     print('Tidak')
...
Tidak
>>>

```

**Gambar 4.16:** Perbedaan respon evaluasi dua variabel berbeda dengan nilai yang sama

```

>>> a='Arya Adhyaksa Waskita'
>>> b=a
>>> if a is b:
...     print('Ya')
... else:
...     print('Tidak')
...
Ya
>>>

```

**Gambar 4.17:** Perbedaan respon evaluasi dua variabel identik dengan nilai yang sama

### 4.3.6 Keanggotaan

Tabel 4.6 menunjukkan operator keanggotaan yang dimiliki Python.



**Tabel 4.6:** Operator keanggotaan di Python

Operator	Fungsi	Contoh
<code>in</code>	<code>True</code> jika nilai tertentu ada dalam obyek	<code>a in b</code>
<code>not in</code>	<code>True</code> jika nilai tertentu tidak ada dalam obyek	<code>a is not b</code>

Perhatikan Program 4.1, di mana angka 7 yang di-*assign* ke variabel `b` akan dicari di dalam list `a` yang berisi 20 angka yang diperoleh secara acak. Jika nilai 7 ada dalam list `a`, program akan mencetak teks 'Ada'. Jika sebaliknya, program akan mencetak teks 'Tidak ada'.

**Program 4.1:** Mendefinisikan fungsi sederhana

```

1 import random
2 a=[]
3 for i in range(20):
4     a.append(random.randint(0,10))
5
6 b=7
7 if b in a:
8     print('Ada')
9 else:
10    print('Tidak ada')
```



## Bab 5

# Mendefinisikan fungsi dan menangani kesalahan

### 5.1 Mendefinisikan fungsi

#### 5.1.1 Fungsi Umum

Mendefinisikan fungsi di Python menggunakan kata kunci `def`, dilanjutkan dengan nama fungsi dan argumen (jika ada). Perhatikan Program 5.1. Fungsi `perkalian` menerima 2 argumen yang dapat dioperasikan secara aritmatika, dalam hal ini perkalian. Kita tidak perlu mendefinisikan jenis variabel `a` dan `b` sebagai argumen di fungsi tersebut. Jika variabel yang diberikan dapat digunakan di dalam fungsi, maka program akan berjalan sesuai yang diharapkan.

**Program 5.1:** Mendefinisikan fungsi sederhana

```
1 def perkalian(a,b):  
2     return a*b  
3  
4 a=5  
5 b=7  
6 print(perkalian(a,b))
```

#### 5.1.2 Lambda

`Lambda` adalah teknik yang tersedia di Python untuk mendefinisikan fungsi yang sederhana. Perhatikan Gambar 5.1. Di gambar tersebut ditunjukkan fungsi `lambda` yang melakukan perkalian terhadap dua bilangan.

```
>>> a=lambda x,y: x*y
>>> print(a(4,5))
20
>>>
```

**Gambar 5.1:** Contoh sederhana fungsi `lambda`

## 5.2 Menangani kesalahan (*exception*)

Coba perhatikan kembali Program 3.7 di sub bab 3.3. Kita dapat mengidentifikasi terjadinya kesalahan selain mengidentifikasi penyebabnya. Karena pada beberapa kasus, penyebab kesalahan sulit diidentifikasi di awal. Perhatikan Program 5.2. Di Program 5.2, jalannya program akan dihentikan ketika nilai `b` yang berperan sebagai pembagi bernilai 0.

**Program 5.2:** Menghentikan perulangan ketika didapati operasi yang tidak valid

```
1 import random
2
3 while True:
4     a=random.randint(0,10)
5     b=random.randint(0,20)
6     try:
7         print(a/b)
8     except:
9         print('Penyebut sama dengan 0')
10        break
```

## Bab 6

# Membuat dan mengelola modul

### 6.1 Pendahuluan

Modul dapat diartikan sebagai fungsi yang dapat dipanggil dari program Python apapun, selama lokasinya diketahui. Modul sangat berguna dalam menyederhanakan struktur aplikasi, sehingga setiap *script* mengerjakan sedikit tugas utama yang saling terkait. Sedangkan tugas lain yang tidak terkait dipisahkan dalam *script* yang berbeda di berkas yang berbeda.

Perhatikan Program 6.1 yang merupakan *script* untuk menghitung nilai faktorial dari bilangan berjenis `int` yang dimasukkan. Faktorial, yang disimbolkan dengan  $n!$ , akan bernilai  $n * n-1 * n-2 * \dots * 1$ .

**Program 6.1:** Menghitung nilai faktorial

```
1 n=input('n=')
2 n=int(n)
3 if n<0:
4     print('Argumen salah')
5 else:
6     f=1
7     if n<2:
8         f=1
9     else:
10        for i in range(1,n+1):
11            f=f*i
12        print(f)
```

Jika perhitungan faktorial lebih dari satu kali, maka akan lebih efisien jika faktorial dibuat dalam fungsi tertentu yang dapat digunakan ulang tanpa menulis ulang. Perhatikan Program 6.2 di mana diperlukan dua kali pemanggilan terhadap fungsi `faktorial`. Dengan menerapkannya sebagai fungsi, kita tidak perlu menulis ulang fungsi `faktorial` (baris ke-1 s/d 8).

**Program 6.2:** Menghitung nilai faktorial yang diterapkan sebagai fungsi

```
1 def faktorial(n):
2     f=1
3     if n<2:
4         f=1
5     else:
6         for i in range(1,n+1):
7             f=f*i
8     print(str(n)+'!='+str(f))
9
10 while True:
11     a=input('a=')
12     a=int(a)
13     if a>=0:
14         break
15
16 while True:
17     b=input('b=')
18     b=int(b)
19     if b>=0:
20         break
21
22 faktorial(a)
23 faktorial(b)
```

## 6.2 Membuat modul

Kita kembali ke kasus faktorial di sub bab 6.1. Asumsikan jika ada beberapa *script* yang membutuhkan fungsi `faktorial` tersebut. Maka, akan ada beberapa *script* yang didalamnya terdefinisi fungsi `faktorial`. Pada kondisi inilah, modul memiliki peran penting. Perhatikan Program 6.3 yang penggunaannya memerlukan *script* seperti Program 6.4.

**Program 6.3:** Fungsi faktorial sebagai modul

```
1 def faktorial(n):
2     f=1
3     if n<2:
4         f=1
5     else:
6         for i in range(1,n+1):
7             f=f*i
8     return f
```

Maksud dari perintah di baris pertama Program 6.4 adalah sebagai berikut.

- **faktorial3** adalah nama file di mana modul faktorial didefinisikan (**faktorial3.py**)
- **faktorail** adalah nama fungsi yang terdefinisi di modul faktorial
- **fk** adalah nama alias yang diberikan sebagai identitas fungsi **faktorial**. Fungsi dengan nama yang relatif panjang sering diberikan nama alias yang lebih singkat, terutama ketika fungsi tersebut sering digunakan

**Program 6.4:** *Script* yang menggunakan modul **faktorial**

```
1 from faktorial3 import faktorial as fk
2 while True:
3     a=input('a=')
4     a=int(a)
5     if a>=0:
6         break
7
8 print(str(a)+'!='+str(fk(a)))
```

Pada kondisi lain, kita mungkin saja perlu melakukan pengujian terhadap modul yang kita bangun. Hal ini dapat disebabkan karena fungsinya kompleks sehingga selalu ada kemungkinan kesalahan dalam tahap pengembangannya. Maka, akan lebih mudah jika kita menyatukan *script* yang bertugas untuk menguji dengan *script* modul. Eksekusi terhadap Program 6.5 yang memanfaatkan modul seperti Program 6.6 akan menghasilkan luaran seperti pada Gambar 6.1.

**Program 6.5:** Modul **faktorial** lengkap dengan fungsi ujinya

```
1 def faktorial(n):
2     f=1
3     if n<2:
4         f=1
5     else:
6         for i in range(1,n+1):
7             f=f*i
8     return f
9
10 while True:
11     a=input('a=')
12     a=int(a)
13     if a>=0:
14         break
15
16 print(str(a)+'!='+str(faktorial(a)))
```

**Program 6.6:** *Script* yang menggunakan modul `faktorial` yang di dalamnya ada fungsi uji

```

1 from faktorial4 import faktorial as fk
2 while True:
3     a=input('a=')
4     a=int(a)
5     if a>=0:
6         break
7
8 print(str(a)+'!='+str(fk(a)))

```

```

a=5
5!=120
a=5
5!=120

```

**Gambar 6.1:** Fungsi faktorial dipanggil dua kali

Gambar 6.1 menunjukkan bahwa fungsi `faktorial` dipanggil dua kali, masing-masing dari fungsi uji di luar modul dan fungsi uji di dalam modul. Untuk mengatasinya, perhatikan Program 6.7. Dengan penambahan baris ke-8 di Program 6.7, fungsi uji yang didefinisikan di modul tidak akan dieksekusi ketika modul tersebut sedang digunakan *script* lain. Program 6.7 juga tampil lebih efisien dalam penulisan karena fungsi rekursif (pemanggilan fungsi itu sendiri) seperti ditunjukkan di baris ke-6.

**Program 6.7:** Modul `faktorial` lengkap dengan fungsi ujinya dan fungsi `main`

```

1 def faktorial(n):
2     f=1
3     if n<2:
4         return 1
5     else:
6         return n * faktorial(n-1)
7
8 if __name__=='__main__':
9     while True:
10        a=input('a=')
11        a=int(a)
12        if a>=0:
13            break
14
15    print(str(a)+'!='+str(faktorial(a)))

```



### 6.3 Jejak pencarian

Modul yang sebelumnya dibuat diletakkan di *directory* yang sama dengan *script* yang akan menggunakannya. Ketika lokasinya berbeda, maka modul tidak dapat digunakan. Agar dapat digunakan oleh *script* di lokasi manapun, kita bisa meletakkan modul di lokasi standar dari modul. Dan untuk mengetahuinya, dapat dilakukan dengan menjalankan dua baris perintah berikut.

```
>>> import sys
>>> sys.path
```

Hasilnya adalah `list` yang berisi lokasi standar dari modul. Kita bisa meletakkan modul kita di lokasi tersebut. Sayangnya, lokasi tersebut umumnya memiliki kewenangan `root` untuk mengaksesnya sehingga kita tidak dapat melakukannya jika kita tidak memiliki hak akses `root`. Untuk kondisi ini, kita dapat menambahkan lokasi modul dengan *directory user* kita. Karena dua perintah di atas menghasilkan `list`, kita bisa melakukan operasi `append` seperti contoh berikut.

```
>>> import sys
>>> sys.path.append('/home/arya/Kuliah/PythonBasic/script/')
```

Dengan demikian, semua *script* Python yang ada di *directory* tersebut dapat digunakan sebagai modul dari lokasi mana saja. Tetapi perlu diingat bahwa proses ini tidak permanen, hanya pada saat *script* tersebut dieksekusi.

Terakhir, sebagai informasi, fungsi faktorial yang disajikan di sini hanya ilustrasi saja. Sejatinya, fungsi tersebut telah tersedia pada modul `math`.



## Bab 7

# Interaksi berkas

### 7.1 Pendahuluan

Lingkup interaksi di sini adalah interaksi dengan berkas teks (ASCII) untuk tujuan membaca, memformat ulang dalam bentuk penyajian di layar maupun disimpan kembali ke berkas teks. Selain teks, maka diperlukan proses tambahan untuk mengubahnya. Hal ini disebabkan karena umumnya data yang dianalisis dalam keilmuan data disimpan dalam bentuk teks dengan format csv (*comma separated value*). Beberapa mungkin disimpan dalam aplikasi *spreadsheet* seperti Microsoft Excel©.

Selain itu, kita mungkin saja terlibat dengan lebih dari satu berkas yang tersebar di *sub directory* yang berbeda. Atau bisa saja perlu menyusun ulang struktur *directory* baru yang lebih mudah dipahami. Satu contoh kasus<sup>1</sup>, data yang berisi citra tumbuhan dari bagian yang berbeda seperti bunga, daun atau buah tidak tersusun berdasarkan bagian-bagian tersebut. Sementara kita memerlukan data tersebut tersusun berdasarkan bagian tumbuhan, bahkan mengikuti struktur *family-genus-species*. Atau contoh lain<sup>2</sup>, di mana dataset terpisah dalam berkas yang berbeda untuk kemudahan proses unduh. Sementara di masing-masing berkas terdapat data dari kelas-kelas yang berbeda. Untuk mengetahui proporsi data dari setiap kelas, kita perlu membaca semua potongan berkas yang tersedia.

### 7.2 Berkas tunggal

Sebagai bahan latihan, kita akan membaca dataset iris<sup>3</sup>. Pembacaan dilakukan baris per baris dan menampilkan isinya. Dataset tersebut berisi informasi tentang dimensi bunga iris, yaitu *sepal length*, *sepal width*, *petal length*, *petal width* yang masing-masing dalam satuan cm. Kolom terakhir merupakan kelas dari 3 jenis bunga iris yang ada pada dataset tersebut, masing-masing Setosa, Versicolor dan Virginica. Data yang ditampilkan harus memiliki format 5 baris yang setiap barisnya adalah

jenis bunga iris ke-i:

---

<sup>1</sup><http://otmedia.lirmm.fr/LifeCLEF/PlantCLEF2017/>

<sup>2</sup>[https://www.unsw.adfa.edu.au/unsw-canberra-cyber/cybersecurity/ADFA-NB15-Datasets/bot\\_iot.php](https://www.unsw.adfa.edu.au/unsw-canberra-cyber/cybersecurity/ADFA-NB15-Datasets/bot_iot.php)

<sup>3</sup><https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data>

```

sepal length = ... cm
sepal width = ... cm
petal length = ... cm
petal width = ... cm

```

Perhatikan Program 7.1. Program tersebut bertujuan untuk membaca data yang disimpan dalam bentuk tabular dan menampilkannya di layar dengan format baru. Simpan Program 7.1 dalam berkas berekstensi `.py` lalu jalankan untuk melihat hasilnya dengan perintah `python nama_file.py`.

**Program 7.1:** Membaca dataset iris dan menampilkan isinya dengan format tertentu

```

1 a=open('iris.data','r')
2 jenis={}
3 total=0
4 for baris in a.readlines():
5     try:
6         element=baris.split(',')
7         kelas=element[4].split('\n')
8         if kelas[0] not in jenis:
9             jenis[kelas[0]]=1
10        else:
11            jenis[kelas[0]]+=1
12        print(kelas[0]+' ke-'+str(jenis[kelas[0]])+':')
13        print('Sepal length: '+element[0]+' cm')
14        print('Sepal width: '+element[1]+' cm')
15        print('Petal length: '+element[2]+' cm')
16        print('Petal width: '+element[3]+' cm')
17        total+=1
18    except:
19        print('End of file')
20 a.close()
21 print('Statistics')
22 for i in jenis.keys():
23     print(i+' : '+str(jenis[i])+' items ('+ '{:4.2f}'.format(jenis[i]/total)+'%)'
        ↪ )

```

Berikut adalah penjelasan perintah pada Program 7.1 berdasarkan urutan barisnya.

1. Baris ke-1: membuat **pointer** ke berkas, yang dalam contoh ini adalah `iris.data` yang menjadi argumen pertama dari fungsi `open`. Sedangkan argumen keduanya adalah `'r'`, menunjukkan mode baca. Untuk mode tulis, gunakan argumen `'w'`.
2. Baris ke-2: menyiapkan variabel *dictionary* yang akan digunakan untuk menyimpan informasi kelas data berikut jumlahnya.

3. Baris ke-3: menyiapkan variabel `int` yang akan digunakan untuk menyimpan informasi total data.
4. Baris ke-4: memulai perulangan untuk membaca berkas per baris melalui perintah `a.readlines()`. Jika hanya diperlukan untuk membaca satu baris saja, gunakan perintah `a.readline()`. Baris yang berhasil dibaca akan disimpan dalam variabel `baris`.
5. Baris ke-5 & 18: penanganan kesalahan yang disebabkan oleh kondisi berkas yang baris terakhirnya kosong sehingga tidak dapat diolah seperti baris lain di atasnya.
6. Baris ke-6: memisahkan baris yang dibaca berdasarkan pembatas (*delimiter*) berupa karakter `,`. Hasilnya disimpan dalam `list` dengan nama `element`.
7. Baris ke-7: membersihkan `element` terakhir dari karakter `\n`, dan menyimpannya dalam `list` dengan nama `kelas`. Nilai dari `kelas` yang akan digunakan disimpan di `kelas[0]`.
8. Baris ke-8 s/d 11: blok kondisi yang ketika nilai kelas belum ada di variabel `jenis`, ia akan ditambahkan sebagai `key` dengan nilai 1. Sedangkan jika sudah menjadi salah satu `key`, maka nilainya ditambah 1.
9. Baris ke-12 s/d 16: menampilkan isi dari baris yang sedang dibaca, yang sebelumnya telah dipisah-pisah ke layar. Sejatinnya, perintah `print` menerima argumen `str`. Jika ada karakter/kata eksplisit akan ditampilkan bersama karakter/kata yang disimpan pada variabel, maka penulisannya perlu dibedakan, dan dihubungkan dengan operator `+`. Dari sini terlihat bahwa operator `+` tidak hanya dapat digunakan dalam operasi aritmatika. Khusus untuk variabel yang akan ditampilkan dengan perintah `print`, tetapi belum dalam bentuk `str` (seperti pada baris ke-12), maka perlu dilakukan transformasi. Perintah yang digunakan adalah `str`, sebagai target jenis variabel yang diperlukan. Jika pada kondisi tertentu, diperlukan untuk mengubah jenis variabel ke `int`, maka digunakan perintah yang sama dengan target jenis variabelnya, dalam hal ini `int`.
10. Baris ke-17: mengakumulasi total data dari berkas yang dibaca.
11. Baris ke-19: mendefinisikan perintah ketika baris yang dibaca tidak dapat diolah dengan perintah yang sama dengan baris lain.
12. Baris ke-20: menghapus `pointer` ke berkas yang telah selesai digunakan.
13. Baris ke-21: mencetak informasi
14. Baris ke-22 & 23: mencetak informasi statistik (jumlah data per kelas dan proporsinya dalam %). Baris ke-22 digunakan untuk membuat perulangan berdasarkan jumlah kelas yang dalam contoh ini adalah jumlah elem dari variabel `jenis`. Sedangkan baris ke-23 digunakan untuk mencetak informasi berikut:
  - nama kelas yang direpresentasikan melalui variabel `i`.
  - jumlah data untuk kelas tertentu yang direpresentasikan melalui variabel `jenis[i]`. Karena jenisnya `int`, maka perlu diubah ke `str`.

- proporsi jumlah data, dilakukan dengan cara melakukan operasi pembagian jumlah data per kelas terhadap total data. Hasilnya ditampilkan dengan format dua angka desimal (:4.2f).

Hasil dari menampilkan informasi statistik tersebut ditampilkan di Gambar 7.1.

```
Iris-setosa: 50 items (0.33%)
Iris-versicolor: 50 items (0.33%)
Iris-virginica: 50 items (0.33%)
```

**Gambar 7.1:** Informasi statistik dataset iris

### 7.3 Berkas jamak

Selanjutnya, sebagai latihan kedua, dataset yang akan digunakan adalah dataset tentang keamanan siber<sup>4</sup>. Untuk seluruh dataset, terdapat 74 berkas yang berbeda. Tetapi, berkas pertama-nya kosong sehingga akan diabaikan dalam pembacaan. Perhatikan Program 7.2.

**Program 7.2:** Menghitung total data untuk setiap kelas

```
1 category={}
2 pre='UNSW_2018_IoT_Botnet_Dataset_'
3 post='.csv'
4 for i in range(2,75):
5     c=0
6     filename=pre+str(i)+post
7     print(filename)
8     a=open(filename,'r')
9     for baris in a.readlines():
10         element=baris.split(',')
11         x=len(element)
12         c=c+1
13         if x>=35:
14             b=element[x-1].split('\n')
15             if b[0] not in category:
16                 category[b[0]]=1
17             else:
18                 category[b[0]]=category[b[0]]+1
19         else:
20             print(c)
21
22     c=0
```

<sup>4</sup>[https://www.unsw.adfa.edu.au/unsw-canberra-cyber/cybersecurity/ADFA-NB15-Datasets/bot\\_iot.php](https://www.unsw.adfa.edu.au/unsw-canberra-cyber/cybersecurity/ADFA-NB15-Datasets/bot_iot.php)

```
23     a.close()
24 print(category)
```

## 7.4 Latihan

### 7.4.1 Berkas tunggal

Program 7.1 dibuat dengan asumsi bahwa dataset memiliki karakteristik sebagai berikut.

- Tidak memiliki informasi kolom (*column header*).
- Memiliki 5 kolom, di mana kolom ke-5 adalah kelas data.

Modifikasi Program 7.1 agar kita tidak perlu membuat asumsi tersebut. Maksudnya, dengan atau tanpa *column header*, tanpa pengetahuan jumlah kolom adalah 5, kita dapat membaca berkas dan menampilkan informasi seperti yang diminta.

### 7.4.2 Berkas jamak

Dataset<sup>5</sup> menyediakan informasi sub kelas. Sebagai contoh, kelas UDP memiliki sub kelas Dos dan DDoS. Sementara Program 7.2 hanya menampilkan informasi jumlah data per kelas. Karena itu, modifikasi Program 7.2 agar dapat secara otomatis menghitung jumlah data per kelas sekaligus per sub kelas dan menampilkannya di layar.

---

<sup>5</sup>[https://www.unsw.adfa.edu.au/unsw-canberra-cyber/cybersecurity/ADFA-NB15-Datasets/bot\\_iot.php](https://www.unsw.adfa.edu.au/unsw-canberra-cyber/cybersecurity/ADFA-NB15-Datasets/bot_iot.php)





## Bab 8

# Interaksi Basisdata

8.1 MySQL

8.2 PostgreSQL

8.3 MongoDB



# Bibliografi

- [Braschler et al., 2019] Braschler, M., Stadelmann, T., and Stockinger, K. (2019). *Applied Data Science: Lessons Learned for the Data-Driven Business*. Springer International Publishing.
- [Dinov, 2018] Dinov, I. (2018). *Data Science and Predictive Analytics: Biomedical and Health Applications using R*. Springer International Publishing.
- [Hunt, 2019] Hunt, J. (2019). *A Beginners Guide to Python 3 Programming*. Springer Publishing Company, Incorporated, 1st edition.
- [Igual et al., 2017] Igual, L., Seguí, S., Vitrià, J., Puertas, E., Radeva, P., Pujol, O., Escalera, S., Dantí, F., and Garrido, L. (2017). *Introduction to Data Science: A Python Approach to Concepts, Techniques and Applications*. Undergraduate Topics in Computer Science. Springer International Publishing.
- [Skiena, 2017] Skiena, S. (2017). *The Data Science Design Manual*. Texts in Computer Science. Springer International Publishing.