
Pemrograman Python

Handbook

Dr. Bambang Purnomosidi D. P.



MAGISTER TEKNOLOGI INFORMASI
STMIK AKAKOM YOGYAKARTA

Daftar Isi

Tentang Buku Ini	1
Mengenai Python	2
Apakah Python Itu?	2
Masalah Apa yang Sesuai untuk Python?	2
Implementasi dan Distribusi Python	3
Algoritma, Pemrograman, Paradigma Pemrograman, dan Python	4
Pengertian Algoritma	4
Pembuatan Algoritma	4
Representasi Algoritma	5
Algoritma dan Paradigma Pemrograman	5
Python dan Implementasi Algoritma	6
Persyaratan Sistem dan Software Pendukung	9
Persyaratan Sistem untuk Python	9
Peranti Pendukung: IDE atau Editor Teks	9
Instalasi Visual Studio Code	10
Instalasi Extension	10
Instalasi Python - Miniconda	11
Mulai Menggunakan Python	18
REPL	18
Coding	19
Dasar-dasar Python	21
Identifikasi / Nama	21
Komentar	23
Variabel dan Tipe Data Dasar	23
Numerik	25
String	25

Operator	26
Indentasi	28
Ekspresi	29
Pengendalian Alur Program	30
if	30
while	30
for	31
Fungsi	33
Apakah Fungsi Itu?	33
Membuat Fungsi	33
Fungsi Dengan Argumen Default	34
Fungsi Dengan Argumen Tidak Pasti	34
Ekspresi / Operator / Fungsi Lambda	35
Struktur Data di Python	37
List	37
Tuple	39
Sets	40
Dictionary	41
Modul dan Conda	42
Modul Standar	42
Modul yang Didefinisikan Pemakai (User Defined Module)	42
pip	43
Conda	44
Operasi I/O	46
Input dari Keyboard	46
Output ke Layar	46
Mengambil Isi File	46
Menangani Error dan Exception	48
OOP di Python	50
Functional Programming di Python	52
Pure Function	52
Iterator	53

Generator	54
Map	54
Reduce	55
Filter	55
Higher Order Function	56
Closure	57
Asynchronous I/O / Concurrent Programming di Python	58

Tentang Buku Ini



Repo ini berisi kode sumber serta buku **Pemrograman Python** dibuat oleh Magister Teknologi Informatika STMIK Akakom. Lisensi semua dokumen pada repo ini adalah Creative Commons Attribution-ShareAlike 4.0 International License - CC-BY-SA 4.0. Secara umum, penggunaan lisensi ini mempunyai implikasi bahwa pengguna materi:

1. Harus memberikan atribusi ke penulis dan sponsor untuk penulisan materi ini (Magister Teknologi Informatika - STMIK Akakom).
2. Boleh menggunakan produk yang ada disini untuk keperluan apapun jika point 1 di atas terpenuhi.
3. Boleh membuat produk derivatif dari produk yang ada disini sepanjang perubahan-perubahan yang dilakukan diberitahukan ke kami dan di-*share* dengan menggunakan lisensi yang sama.

Untuk penggunaan selain ketentuan tersebut, silahkan menghubungi:

```
1 Magister Teknologi Informasi
2 STMIK Akakom
3 Jl. Raya Janti Karangjambe no 143
4 Yogyakarta 55198
5 Indonesia
6 Phone: 0274 486664
7 https://pascasarjana.akakom.ac.id
8 info.mti@akakom.ac.id
```

Mengenal Python

Apakah Python Itu?

Python adalah spesifikasi bahasa pemrograman serta peranti penerjemah (*interpreter*) untuk menjalankan / mengeksekusi *source code* yang dibuat menggunakan bahasa pemrograman Python tersebut. Python dibuat pertama kali oleh **Guido van Rossum** dan saat ini dikembangkan oleh komunitas di bawah kendali PSF (Python Software Foundation - <https://www.python.org/psf/>). Untuk selanjutnya, istilah Python akan mengacu pada spesifikasi serta software untuk interpreter Python tersebut.

Masalah Apa yang Sesuai untuk Python?

Python digunakan untuk pemrograman umum (bisa digunakan untuk berbagai domain masalah), bertipe dinamis, tidak perlu dikompilasi (*interpreted*), mendukung berbagai paradigma pemrograman (OOP, *functional*, *procedural*, imperatif) serta bisa digunakan di berbagai platform (Windows, Linux, MacOS, FreeBSD, NetBSD, dan lain-lain). Ruang lingkup aplikasi yang bisa dibuat menggunakan Python pada dasarnya meliputi banyak hal kecuali *machine low level* (*interfacing* dengan *hardware*, membuat sistem operasi, dan sejenisnya). Meskipun kadang Python digunakan untuk peranti pengembangan yang terkait dengan akses low level, biasanya Python hanya merupakan peranti level atas - akses ke peranti keras dibuat menggunakan C/C++/Rust dan dikompilasi menjadi modul Python. Python juga tidak cocok digunakan untuk pembuatan aplikasi mobile phone. Untuk memberikan gambaran masalah apa saja yang bisa diselesaikan menggunakan Python, silahkan melihat pada daftar kisah sukses Python di <https://www.python.org/about/success/>

Beberapa domain aplikasi Python antara lain adalah sebagai berikut:

1. Web dan Internet: Django, Flask, TurboGears, dan lain-lain
2. Komputasi sains, kecerdasan buatan, dan riset: PyTorch, Keras, Tensorflow, SciPy, scikit-learn, spacy dan NLTK (untuk NLP), numpy, Bokeh, Dash, dan lain-lain
3. Pendidikan: Python sering digunakan sebagai bahasa pemrograman untuk mengajarkan pemrograman

4. Pembuatan Game: pygame, dan lain-lain
5. GUI (Tkinter, wxpython, PyQt, PyGTK, dan lain-lain) maupun TUI (python-ncurses, dan lain-lain)
6. Bahasa pemrograman untuk membuat tools di *software development*.
7. Aplikasi bisnis: Odoo untuk ERP, dan lain-lain.

Implementasi dan Distribusi Python

Secara umum, software Python biasanya bisa diambil dari <https://www.python.org> meskipun beberapa perusahaan maupun komunitas developer juga membuat distribusi Python maupun versi interpreter Python untuk platform tertentu. Python dari situs web tersebut dikenal dengan istilah **CPython** dan merupakan *reference implementation* dari spesifikasi Python. Beberapa distribusi atau implementasi Python lainnya:

- **Jython** (Python di JVM).
- **Pyston** (Python yang diimplementasikan dengan teknik JIT).
- **RustPython** (Python yang diimplementasikan menggunakan bahasa pemrograman Rust)
- **IronPython** (Python di .NET)
- **Stackless** (Python dengan microthreads - threads yang tidak dikelola oleh OS, tetapi dikelola oleh Stackless).
- **MicroPython** (Python untuk microcontroller)
- **CircuitPython** (turunan dari MicroPython untuk siswa dan pemula).
- **PyPy** (Python JIT Compiler)
- **Anaconda** (Python standar yang sudah menyertakan conda).
- **Intel Distribution for Python**, bisa diperoleh di situs Intel (<https://software.intel.com/en-us/python-distribution>).

Materi di *handbook* ini menggunakan standar Python (CPython) serta Anaconda / Miniconda / conda. Saat ini, versi Python ada 2: versi 2.x dan versi 3.x. Keduanya tidak kompatibel. Materi ini menggunakan versi 3.x. Untuk proyek pengembangan software yang baru, disarankan untuk menggunakan Python 3 karena Python 2 sudah akan memasuki fase EOL **End Of Life** pada bulan Januari 2020. Status mengenai Python 2.0 bisa dilihat pada URL berikut: <https://www.python.org/dev/peps/pep-0373/>.

Algoritma, Pemrograman, Paradigma Pemrograman, dan Python

Pengertian Algoritma

Istilah *Algoritma* berasal dari nama matematikawan abad 9, yaitu * Muḥammad ibn Mūsā al-Khwārizmī* (pada bahasa latin, dipanggil dengan nama *Algoritmi*). Algoritma merupakan spesifikasi langkah dalam menyelesaikan suatu masalah.

Pada intinya, pemrograman diperlukan untuk membuat solusi terhadap suatu permasalahan tertentu. Seorang pemrograman merupakan seseorang yang bertugas untuk membuat program. Proses pembuatan program tersebut sering disebut juga dengan *coding*. Proses *coding* tersebut pada dasarnya adalah proses untuk menganalisis masalah, mengabstraksi masalah, mewujudkan abstraksi tersebut dalam bentuk *algoritma*, mengimplementasikan algoritma tersebut dengan suatu bahasa pemrograman tertentu, menguji implementasi tersebut, serta melakukan proses *debugging* saat terjadi kesalahan. Pada banyak kasus, sering juga tidak hanya sampai disitu, tetapi juga sampai ke analisis algoritma untuk mencari algoritma yang paling optimum.

Pembuatan Algoritma

Untuk membangun suatu program dengan melibatkan algoritma, biasanya diperlukan langkah-langkah berikut ini:

1. Definisi masalah
2. Abstraksi masalah dalam bentuk model
3. Spesifikasikan dan rancang langkah-langkah untuk menyelesaikan model tersebut
4. Implementasikan rancangan tersebut dalam *source code*.
5. Uji kebenaran algoritma tersebut, biasanya dengan membuat test yang terdiri atas masukan serta keluaran yang dikehendaki.
6. *Debug* atau cari kesalahan jika terdapat kesalahan. Betulkan kesalahan tersebut.
7. Jika sudah berjalan, cari lagi algoritma yang paling baik - proses ini disebut *optimasi* dan bisa dilakukan dengan analisis algoritma.

8. Testing program
9. Buat dokumentasi.

Representasi Algoritma

Jika digambarkan, algoritma sering diwujudkan dalam bentuk alur program (*flowchart*) maupun dalam bentuk *pseudocode*. Pembuatan *flowchart* maupun *pseudocode* ini seringkali digunakan dalam dunia akademik dan relatif jarang digunakan di industri. Meskipun demikian, bisa dipastikan bahwa setiap pemrogram - tidak peduli di industri maupun dunia akademik - akan selalu menggunakan algoritma.

Contoh algoritma untuk mencari nilai terbesar dari sekumpulan angka adalah sebagai berikut (diambil dari <https://en.wikipedia.org/wiki/Algorithm>):

```
1 Algorithm LargestNumber
2   Input: A list of numbers L.
3   Output: The largest number in the list L.
4   if L.size = 0 return null
5   largest ← L[0]
6   for each item in L, do
7     if item > largest, then
8       largest ← item
9   return largest
```

Algoritma dan Paradigma Pemrograman

Paradigma pemrograman merupakan cara pandang dari pemrogram untuk mengimplementasikan algoritma serta menyelesaikan suatu masalah tertentu dalam pemrograman. Perbedaan antara berbagai paradigma ini terutama dalam hal implementasi algoritma serta komponen dan unit terkecil untuk *reusable code*. Pada umumnya, saat ini ada beberapa paradigma pemrograman yang banyak diimplementasikan dalam bahasa pemrograman:

1. **Prosedural**: implementasi algoritma dalam bentuk langkah-langkah prosedural dan unit terkecil diabstraksi dalam bentuk *procedure* (tidak menghasilkan nilai kembalian) maupun *function* (menghasilkan nilai kembalian).
2. **Fungsional**: implementasi algoritma dalam bentuk langkah-langkah dengan ketetapan prinsip-prinsip matematika dan unit terkecil diimplementasikan dalam bentuk *function*.

3. **Object-oriented:** implementasi algoritma dalam bentuk langkah-langkah dengan menggunakan pola pikir natural dengan prinsip-prinsip obyek dan unit terkecil diimplementasikan dalam bentuk kelas (*class*) serta pembuatan *instance* atau obyek yang saling bekerjasama dan berinteraksi.

Python mendukung ketiga pola pikir dan paradigma pemrograman tersebut. Secara umum, paradigma pertama (*prosedural*) relatif lebih mudah dipahami daripada dua paradigma lainnya. Dalam proses pembelajaran, biasanya paradigma *prosedural* ini yang lebih dulu dipelajari.

Python dan Implementasi Algoritma

Python merupakan bahasa pemrograman yang mempunyai sintaksis atau tata bahasa yang menyerupai *pseudocode* sehingga relatif mudah untuk dipelajari (khususnya jika bahasa Inggris tidak menjadi masalah). Secara umum, implementasi Python dari algoritma bisa diwujudkan dalam proses berikut ini:

1. Definisi masalah
2. Rancang algoritma
3. Gnakan sintaks Python untuk mengimplementasikan langkah per langkah dari algoritma.

Contoh dari proses tersebut adalah sebagai berikut:

1. Definisi msaalah

```
1 Menentukan nilai nilai terbesar antara kedua angka
```

2. Rancang algoritma

```
1 0. mulai
2 1. meminta masukan angka pertama
3 2. meminta masukan angka kedua
4 3. pemeriksaan dan penentuan hasil:
5
6 Jika angka pertama lebih besar daripada angka kedua:
7     cetak angka pertama sebagai angka terbesar
8 Selain itu, jika angka pertama lebih kecil daripada angka kedua
9     cetak angka kedua sebagai angka terbesar
10 Selain itu, jika angka pertama sama dengan angka kedua
11     cetak pemberitahuan bahwa tidak ada angka terbesar karena sama
12 Selain itu,
13     cetak pemberitahuan bahwa ada kesalahan masukan
```

```
14
15 4. selesai
```

3. Implementasikan dalam Python

Setelah memodelkan penyelesaian dalam bentuk algoritmis berupa langkah-langkah, cari perintah-perintah Python yang bisa digunakan untuk menyelesaikan langkah-langkah tersebut:

1. meminta masukan: *input()*
2. memeriksa dan menentukan hasil: *if ... elif ... else*
3. mencetak: *print*

Cara mengetahui perintah-perintah tersebut biasanya dengan melihat pada manual dokumentasi dari bahasa pemrograman. Manual bahasa pemrograman biasanya minimal terdiri atas 2 bagian:

1. Dokumentasi sintaksis: konstruksi bahasa pemrograman (variabel, function, class, *statement* pengendali, dan lain-lain).
2. Dokumentasi pustaka / perintah standar: perintah-perintah siap pakai (menampilkan tulisan / *print*, akses ke sistem operasi, pemrosesan file, dan lain-lain).

Setelah itu, implementasikan algoritma tersebut:

```
1 angka1 = input("Masukkan angka pertama: ")
2 angka2 = input("Masukkan angka kedua: ")
3
4 if angka1 > angka2:
5     print("Nilai terbesar: ", angka1)
6 elif angka1 < angka2:
7     print("Nilai terbesar: ", angka2)
8 elif angka1 == angka2:
9     print("Kedua angka sama, tidak ada yang terbesar")
10 else:
11     print("Ada masalah dengan angka masukan anda")
```

Untuk menjalankan:

```
1 $ python max.py
2 Masukkan angka pertama: 3
3 Masukkan angka kedua: 4
4 Nilai terbesar: 4
5 $ python max.py
6 Masukkan angka pertama: 4
7 Masukkan angka kedua: 3
8 Nilai terbesar: 4
```

```
9 $ python max.py
10 Masukkan angka pertama: 5
11 Masukkan angka kedua: 5
12 Kedua angka sama, tidak ada yang terbesar
13 $
```

Persyaratan Sistem dan Software Pendukung

Persyaratan Sistem untuk Python

Persyaratan sistem untuk menggunakan Python tidak berat, cukup dengan RAM 2 GB dan spesifikasi laptop / PC biasa (Intel/AMD processor) sudah bisa digunakan untuk menjalankan Python. Setelah itu, perhatikan kebutuhan terkait masalah yang ingin diselesaikan. Jika hanya digunakan untuk materi pelajaran Python biasa, maka spesifikasi di atas sudah cukup. Jika melibatkan proses pemrograman dan pembuatan software yang relatif serius dan menggunakan sumber daya yang besar (misal mengolah data sampai dengan jutaan rekaman, proses *machine learning* dengan materi *training* yang besar, dan sejenisnya), maka spesifikasi di atas tentu saja tidak cukup. Untuk *machine learning* bahkan diperlukan peranti dengan GPU dengan memory serta processor sangat besar.

Peranti Pendukung: IDE atau Editor Teks

Peranti pendukung untuk Python yang paling utama adalah IDE (*Integrated Development Environment*). IDE merupakan software yang berisi berbagai peranti untuk membantu proses pengembangan aplikasi dan terdiri atas komponen yang lengkap, mulai dari komponen editor untuk mengetikkan program sampai dengan *debugger* untuk pencarian kesalahan maupun *profiler* untuk mengoptimasi program yang dibuat. Sebenarnya, komponen minimal dari peranti pendukung yang harus ada adalah editor teks (*text editor*). Editor teks digunakan untuk menuliskan program, biasanya mempunyai fasilitas *syntax highlighting*. Beberapa editor teks memang sudah menyertakan fasilitas tersebut sebagai fasilitas standar (misal Vim di Linux, Notepad++ di Windows).

Jika editor teks biasa dirasa tidak mencukupi, maka bisa digunakan IDE. Ada banyak IDE yang bisa digunakan di Python, tetapi di buku ini akan digunakan Visual Studio Code yang bisa diperoleh di <https://code.visualstudio.com/>. Secara default, VSCode tidak mempunyai dukungan IDE untuk Python, sehingga perlu menginstall *extension* untuk Python. VSCode mensyaratkan RAM minimum 4 GB.

Instalasi Visual Studio Code

VSCode tersedia untuk setidaknya 3 platform: Windows, Mac, dan Linux. Download VSCode dari <https://code.visualstudio.com/Download> dan ekstrak file tersebut di lokasi pilihan masing-masing. Setelah itu, jalankan dengan memanggil **code**. Pada semua platform, VSCode akan mempunyai tampilan yang sama:

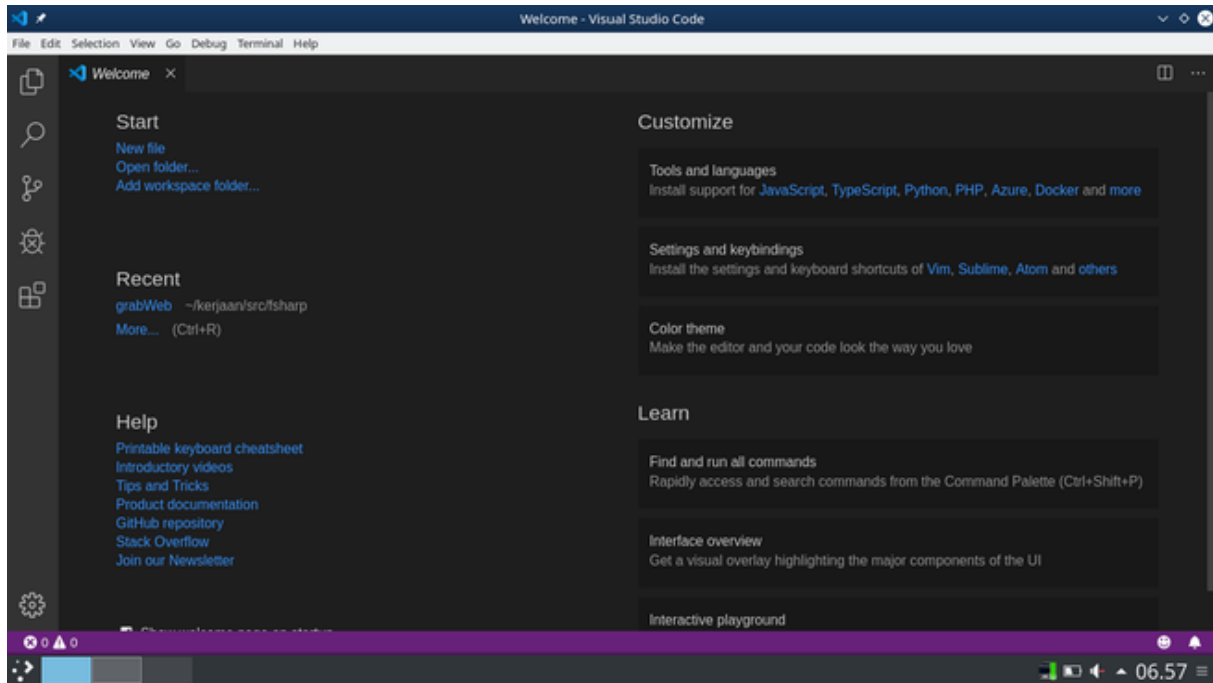


Figure 0.1: VSCode

Pada posisi ini, semua platform mempunyai cara pengoperasian yang sama.

Instalasi Extension

Extension Python untuk VSCode tersedia di *marketplace* pada URL <https://marketplace.visualstudio.com/items?itemName=python.python>. Untuk melakukan instalasi extension, buka *Extensions* di VSCode dengan klik tombol *Extensions* pada sisi kiri atau tekan langsung *Ctrl-Shift-X*. Setelah itu, cari dan install. Pilih Extension dari Microsoft kemudian klik *Install*. Bisa juga menggunakan Quick Open atau *Ctrl-P* kemudian memasukkan perintah berikut:

```
1 ext install ms-python.python
```

Instalasi Python - Miniconda

Pada umumnya, komputer yang diinstall Linux sudah mempunyai Python. Meskipun demikian, seringkali versi yang ada masih versi lama. Proses uninstall untuk kasus tersebut juga tidak memungkinkan karena biasanya akan membuat banyak software lainnya menjadi tidak bisa digunakan di komputer tersebut (broken). Miniconda memungkinkan kita menginstall versi stabil maupun versi lainnya. Distribusi Miniconda bisa diperoleh di <https://conda.io/miniconda.html>. Contoh instalasi akan diberikan untuk Linux 64 bit dan Python versi 3. Download pada lokasi di atas, setelah itu jalankan file tersebut setelah di - chmod:

```
1  » chmod +x Miniconda3-latest-Linux-x86_64.sh
2  » ./Miniconda3-latest-Linux-x86_64.sh
3
4  Welcome to Miniconda3 4.5.11
5
6  In order to continue the installation process, please review the
   license
7  agreement.
8  Please, press ENTER to continue
9  >>>
10 =====
11 Miniconda End User License Agreement
12 =====
13
14 Copyright 2015, Anaconda, Inc.
15
16 All rights reserved under the 3-clause BSD License:
17
18 Redistribution and use in source and binary forms, with or without
   modification, are permitted provided that the following conditions
   ar
19 e met:
20
21 * Redistributions of source code must retain the above copyright
   notice, this list of conditions and the following disclaimer.
22 * Redistributions in binary form must reproduce the above copyright
```

```
notice, this list of conditions and the following disclaimer in th
23 e documentation and/or other materials provided with the distribution.
24 * Neither the name of Anaconda, Inc. ("Anaconda, Inc.") nor the names
    of its contributors may be used to endorse or promote products d
25 erived from this software without specific prior written permission.
26
27 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS
    IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LI
28 MITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
    PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL ANACONDA, INC
29 . BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY,
    OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREME
30 NT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
    BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILI
31 TY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
    NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
    SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
32
33
34 Notice of Third Party Software Licenses
35 =====
36
37 Miniconda contains open source software packages from third parties.
    These are available on an "as is" basis and subject to their indivi
38 dual license agreements. These licenses are available in Anaconda
    Distribution or at http://docs.anaconda.com/anaconda/pkg-docs. Any
    bin
39 ary packages of these third party tools you obtain via Anaconda
    Distribution are subject to their individual licenses as well as the
    Ana
40 conda license. Anaconda, Inc. reserves the right to change which third
    party tools are provided in Miniconda.
41
42 Cryptography Notice
43 =====
44
45 This distribution includes cryptographic software. The country in which
    you currently reside may have restrictions on the import, posses
46 sion, use, and/or re-export to another country, of encryption software.
    BEFORE using any encryption software, please check your country'
47 s laws, regulations and policies concerning the import, possession, or
    use, and re-export of encryption software, to see if this is perm
48 itted. See the Wassenaar Arrangement http://www.wassenaar.org/ for more
```



```
information.
49
50 Anaconda, Inc. has self-classified this software as Export Commodity
    Control Number (ECCN) 5D992b, which includes mass market informatio
51 n security software using or performing cryptographic functions with
    asymmetric algorithms. No license is required for export of this so
52 ftware to non-embargoed countries. In addition, the Intel(TM) Math
    Kernel Library contained in Anaconda, Inc.'s software is classified
    b
53 y Intel(TM) as ECCN 5D992b with no license required for export to non-
    embargoed countries.
54
55 The following packages are included in this distribution that relate to
    cryptography:
56
57 openssl
58     The OpenSSL Project is a collaborative effort to develop a robust,
        commercial-grade, full-featured, and Open Source toolkit
        implemen
59 ting the Transport Layer Security (TLS) and Secure Sockets Layer (SSL)
        protocols as well as a full-strength general purpose cryptography
60 library.
61
62 pycrypto
63     A collection of both secure hash functions (such as SHA256 and
        RIPEMD160), and various encryption algorithms (AES, DES, RSA,
        ElGamal
64 , etc.).
65
66 pyopenssl
67     A thin Python wrapper around (a subset of) the OpenSSL library.
68
69 kerberos (krb5, non-Windows platforms)
70     A network authentication protocol designed to provide strong
        authentication for client/server applications by using secret-
        key cryptography.
71
72 cryptography
73     A Python library which exposes cryptographic recipes and primitives
        .
74
75
76 Do you accept the license terms? [yes|no]
```

```
77 [no] >>> yes
78
79 Miniconda3 will now be installed into this location:
80 /home/bpdp/miniconda3
81
82 - Press ENTER to confirm the location
83 - Press CTRL-C to abort the installation
84 - Or specify a different location below
85
86 [/home/bpdp/miniconda3] >>> /opt/software/python-dev-tools/miniconda3
87 PREFIX=/opt/software/python-dev-tools/miniconda3
88 installing: python-3.7.0-hc3d631a_0 ...
89 Python 3.7.0
90 installing: ca-certificates-2018.03.07-0 ...
91 installing: conda-env-2.6.0-1 ...
92 installing: libgcc-ng-8.2.0-hdf63c60_1 ...
93 installing: libstdcxx-ng-8.2.0-hdf63c60_1 ...
94 installing: libffi-3.2.1-hd88cf55_4 ...
95 installing: ncurses-6.1-hf484d3e_0 ...
96 installing: openssl-1.0.2p-h14c3975_0 ...
97 installing: xz-5.2.4-h14c3975_4 ...
98 installing: yaml-0.1.7-had09818_2 ...
99 installing: zlib-1.2.11-ha838bed_2 ...
100 installing: libedit-3.1.20170329-h6b74fdf_2 ...
101 installing: readline-7.0-h7b6447c_5 ...
102 installing: tk-8.6.8-hbc83047_0 ...
103 installing: sqlite-3.24.0-h84994c4_0 ...
104 installing: asn1crypto-0.24.0-py37_0 ...
105 installing: certifi-2018.8.24-py37_1 ...
106 installing: chardet-3.0.4-py37_1 ...
107 installing: idna-2.7-py37_0 ...
108 installing: pycosat-0.6.3-py37h14c3975_0 ...
109 installing: pycparser-2.18-py37_1 ...
110 installing: pysocks-1.6.8-py37_0 ...
111 installing: ruamel_yaml-0.15.46-py37h14c3975_0 ...
112 installing: six-1.11.0-py37_1 ...
113 installing: cffi-1.11.5-py37he75722e_1 ...
114 installing: setuptools-40.2.0-py37_0 ...
115 installing: cryptography-2.3.1-py37hc365091_0 ...
116 installing: wheel-0.31.1-py37_0 ...
117 installing: pip-10.0.1-py37_0 ...
118 installing: pyopenssl-18.0.0-py37_0 ...
119 installing: urllib3-1.23-py37_0 ...
```

```
120 installing: requests-2.19.1-py37_0 ...
121 installing: conda-4.5.11-py37_0 ...
122 installation finished.
123 Do you wish the installer to prepend the Miniconda3 install location
124 to PATH in your /home/bpdp/.bashrc ? [yes|no]
125 [no] >>> yes
126
127 Appending source /opt/software/python-dev-tools/miniconda3/bin/activate
    to /home/bpdp/.bashrc
128 A backup will be made to: /home/bpdp/.bashrc-miniconda3.bak
129
130
131 For this change to become active, you have to open a new terminal.
132
133 Thank you for installing Miniconda3!
134 »
```

Setelah itu, atur environment variable (variabel lingkungan) pada file dan source file tersebut setiap kali ingin menjalankan Python dari Anaconda. Jika menggunakan shell fish:

```
1 » cat env-fish/anaconda/miniconda3
2 set -x PATH /opt/software/python-dev-tools/miniconda3/bin $PATH
3 » source env-fish/anaconda/miniconda3
4 » python
5 Python 3.7.0 (default, Jun 28 2018, 13:15:42)
6 [GCC 7.2.0] :: Anaconda, Inc. on linux
7 Type "help", "copyright", "credits" or "license" for more information.
8 >>>
9 » conda
10 usage: conda [-h] [-V] command ...
11
12 conda is a tool for managing and deploying applications, environments
    and packages.
13
14 Options:
15
16 positional arguments:
17   command
18     clean          Remove unused packages and caches.
19     config          Modify configuration values in .condarc. This is
        modeled
20                   after the git config command. Writes to the user .
        condarc
```

```
21         file (/home/bpdp/.condarc) by default.
22     create      Create a new conda environment from a list of
        specified
23         packages.
24     help       Displays a list of available conda commands and their
        help
25         strings.
26     info        Display information about current conda install.
27     install     Installs a list of packages into a specified conda
28         environment.
29     list        List linked packages in a conda environment.
30     package     Low-level conda package utility. (EXPERIMENTAL)
31     remove      Remove a list of packages from a specified conda
        environment.
32     uninstall   Alias for conda remove. See conda remove --help.
33     search      Search for packages and display associated information
        . The
34         input is a MatchSpec, a query language for conda
        packages.
35         See examples below.
36     update      Updates conda packages to the latest compatible
        version. This
37         command accepts a list of package names and updates
        them to
38         the latest versions that are compatible with all other
39         packages in the environment. Conda attempts to install
        the
40         newest versions of the requested packages. To
        accomplish
41         this, it may update some packages that are already
        installed,
42         or install additional packages. To prevent existing
        packages
43         from updating, use the --no-update-deps option. This
        may
44         force conda to install older versions of the requested
45         packages, and it does not prevent additional
        dependency
46         packages from being installed. If you wish to skip
        dependency
47         checking altogether, use the '--force' option. This
        may
48         result in an environment with incompatible packages,
```

```
so this
49         option must be used with great caution.
50     upgrade    Alias for conda update. See conda update --help.
51
52 optional arguments:
53   -h, --help      Show this help message and exit.
54   -V, --version  Show the conda version number and exit.
55
56 conda commands available from other packages:
57   env
58   bdpd at archer1 in ~
59   »
```

Jika menggunakan shell Bash:

```
1  » cat env-bash/anaconda/miniconda3
2  export PATH=/opt/software/python-dev-tools/miniconda3/bin:$PATH
3  » source env-bash/anaconda/miniconda3
4  » python
5  Python 3.7.0 (default, Jun 28 2018, 13:15:42)
6  [GCC 7.2.0] :: Anaconda, Inc. on linux
7  Type "help", "copyright", "credits" or "license" for more information.
8  >>>
```

Jika menggunakan Windows, instalasi dengan Windows installer sudah melakukan berbagai konfigurasi sehingga bisa menjalankan langsung dari command prompt. Jika langkah-langkah di atas bisa dilakukan, maka Python dan conda sudah terinstall. Python akan digunakan untuk menjalankan source code dalam bahasa pemrograman Python, sementara conda akan digunakan untuk mengelola paket serta variabel lingkungan.

Mulai Menggunakan Python

Python bisa digunakan dengan 2 cara:

1. REPL (Read-Eval-Print-Loop)
2. Coding - membuat source code dalam Python dan eksekusi /jalankan menggunakan interpreter Python.

REPL

1 REPL digunakan untuk keperluan mencoba potongan source code:

```
1 » python
2 Python 3.7.0 (default, Oct  9 2018, 10:31:47)
3 [GCC 7.3.0] :: Anaconda, Inc. on linux
4 Type "help", "copyright", "credits" or "license" for more information.
5 >>> help
6 Type help() for interactive help, or help(object) for help about object
7
8
9 Welcome to Python 3.7's help utility!
10
11 If this is your first time using Python, you should definitely check
12   out
13   the tutorial on the Internet at https://docs.python.org/3.7/tutorial/.
14
15 Enter the name of any module, keyword, or topic to get help on writing
16 Python programs and using Python modules. To quit this help utility
17   and
18   return to the interpreter, just type "quit".
19
20 To get a list of available modules, keywords, symbols, or topics, type
21 "modules", "keywords", "symbols", or "topics". Each module also comes
22 with a one-line summary of what it does; to list the modules whose name
```

```
21 or summary contain a given string such as "spam", type "modules spam".
22
23 help> for
24
25 Related help topics: break, continue, while
26
27 help> while
28
29 Related help topics: break, continue, if, TRUTHVALUE
30
31 help>
32 You are now leaving help and returning to the Python interpreter.
33 If you want to ask for help on a particular object directly from the
34 interpreter, you can type "help(object)". Executing "help('string')"
35 has the same effect as typing a particular string at the help> prompt.
36 >>>
```

Contoh penggunaan:

```
1 >>> for i in range(5):
2 ...     print(i)
3 ...
4 0
5 1
6 2
7 3
8 4
9 5
10 >>> a = 20
11 >>> a
12 20
13 >>>
```

Pada REPL, a akan menghasilkan angka 20 (nilai dari a) karena sifat P di dalam REPL yang akan menampilkan (Print) hasil evaluasi. Di dalam script, harus eksplisit menggunakan perintah print jika ingin menampilkan sesuatu. Setelah selesai dan ingin keluar dari REPL, tekan Ctrl-D.

Coding

Jika software yang akan dibuat mulai besar dan serius, maka REPL tidak cocok digunakan lagi. Untuk keperluan ini, tulis source code dalam bahasa pemrograman Python, kemudian jalankan den-

gan menggunakan interpreter Python. Penulisan source code biasanya memerlukan suatu software khusus, sebagai contoh, programmer Python bisa menggunakan vim, Emacs, Visual Studio Code, dan lain-lain. Untuk keperluan ini, ada 3 cara:

Cara 1

```
1 # hello.py
2 print('Halo')
3 » python hello.py
4 Halo
5 »
```

Cara 2

```
1 # hello2.py
2 #!/usr/bin/env python
3
4 print('Halo')
5 » chmod +x hello2.py
6 » ./hello2.py
7 Halo
8 »
```

Cara 3

```
1 #hello3.py
2 #!/opt/software/python-dev-tools/miniconda3/bin/python
3
4 print('Halo')
5 » chmod +x hello2.py
6 » ./hello3.py
7 Halo
8 »
```


Dasar-dasar Python

Identifier / Nama

Saat memprogram menggunakan Python, seorang programmer harus berurusan dengan nama / identifier, misalnya nama variabel, nama kelas, dan lain-lain. Berikut adalah ketentuan nama yang sah di Python:

- Bukan merupakan kata kunci / keyword di Python
- Membedakan huruf besar dan kecil
- Tidak boleh dimulai dengan digit (0-9)
- Digit hanya boleh setelah karakter pertama
- Boleh huruf besar atau kecil
- Karakter yang diperbolehkan: underscore (_).
- Tidak boleh menggunakan karakter-karakter yang sudah ada di Python untuk keperluan tertentu (misalnya asterisk / + -).

Meskipun tidak ada aturan, di kalangan pemrogram Python, biasanya digunakan konvensi berikut ini:

- Nama modul harus huruf kecil, jika diperlukan bisa menggunakan underscore.
- Nama variabel dan nama fungsi / method harus huruf kecil dan menggunakan underscore jika terdiri atas 2 kata atau lebih.
- Nama kelas harus CamelCase.
- Konstanta harus huruf besar semua.

Contoh:

```
1 modul>NamaKelas.nama_method
```

Beberapa keyword dari Python adalah:

- and
- def
- False

- import
- not
- True
- as
- del
- finally
- in
- or
- try
- assert
- elif
- for
- is
- pass
- while
- break
- else
- from
- lambda
- print
- with
- class
- except
- global
- None
- raise
- yield
- continue
- exec
- if
- non
- local
- return

Untuk mengetahui apakah suatu string merupakan keyword Python, gunakan:

```
1 >>> import keyword
2 >>> keyword.iskeyword('with')
3 True
```

```
4 >>> keyword.iskeyword('for')
5 True
6 >>> keyword.iskeyword('x')
7 False
```

Komentar

Baris(-baris) tertentu dalam kode sumber Python biasanya digunakan untuk membuat keterangan atau dokumentasi. Supaya tidak dijalankan, maka harus dimasukkan dalam komentar:

```
1 # komentar satu baris penuh
2 print('abc) # komentar mulai kolom tertentu"""
3 komentar
4 Lebih dari
5 satu baris"""
```

Variabel dan Tipe Data Dasar

Variabel adalah nama yang digunakan untuk menyimpan suatu nilai. Nama ini nantinya akan digunakan dalam proses-proses program selanjutnya. Perintah umumnya adalah:

```
1 Nama_var = nilai
2
3 var01 = 20
4 var_02 = 30
5 nama_var = 'Satu dua tiga'
6
7 print(var01)
8 print(var_02)
9 print(nama_var)
10
11 # ini salah
12 var 01 = 21
```

Bentuk penugasan (pengisian variabel) lainnya:

```
1 >>> var1 = var2 = var3 = 4
2 >>> var1
3 4
4 >>> var2
```

```
5 4
6 >>> var3
7 4
8 >>> v1, v2, v3 = 'isi 1', 20, 43
9 >>> v1
10 'isi 1'
11 >>> v2
12 20
13 >>> v3
14 43
15 >>>
```

Python adalah bahasa pemrograman yang termasuk dalam kategori dynamic typing, artinya tipe data suatu variabel nanti bisa berubah / bersifat dinamis, berbeda dari apa yang telah dideklarasikan pada awalnya:

```
1 >>> var1 = 143
2 >>> var2 = var1 + 2
3 >>> var2
4 145
5 >>> var1 = 'Wabi Teknologi'
6 >>> var2 = var1 + 2
7 Traceback (most recent call last):
8   File "<stdin>", line 1, in <module>
9   TypeError: can only concatenate str (not "int") to str
10 >>>
```

Variabel juga bisa dihapus:

```
1 >>> a = 10
2 >>> a
3 10
4 >>> del a
5 >>> a
6 Traceback (most recent call last):
7   File "<stdin>", line 1, in <module>
8   NameError: name 'a' is not defined
9 >>>
```

Ada beberapa tipe data dasar yang bisa disimpan oleh variabel.

Numerik

Ada 3 tipe angka: integer (bilangan bulat), float (bilangan pecahan), serta complex (bilangan kompleks).

```
1 >>> sys.float_info
2 sys.float_info(max=1.7976931348623157e+308, max_exp=1024, max_10_exp
   =308, min=2.2250738585072014e-308, min_exp=-1021, min_10_exp=-307,
   dig=15, mant_dig=53, epsilon=2.220446049250313e-16, radix=2, rounds
   =1)
3 >>> sys.int_info
4 sys.int_info(bits_per_digit=30, sizeof_digit=4)
5 >>> sys.maxsize
6 9223372036854775807
7 >>>
```

Bilangan kompleks:

```
1 x = 6
2 y = 4
3
4 z = complex(x,y);
5
6 print ("Bagian bilangan riil: ", z.real)
7 print ("Bagian imajiner dari: ", z.imag)
```

String

String digunakan untuk menyimpan data karakter / huruf / angka yang tidak dimaksudkan untuk operasi matematika.

```
1 str1 = 'string 1'
2 str2 = "string 2"
3 str3 = """ini baris pertama
4 ini baris kedua
5 ini baris ketiga
6 """
7 print(str1)
8 print(str2)
9 print(str3)
10 print(str1[3])
```

Operator

Operator merupakan simbol yang digunakan untuk melakukan suatu operasi terhadap satu atau lebih operand, misal:

```
1 1 + 3
```

adalah simbol untuk melakukan operasi penjumlahan terhadap 2 operand yaitu 1 dan 3. Ada beberapa tipe operator di Python. Potongan source code di bawah ini memperlihatkan jenis dan penggunaannya.

```
1 print('Operator Aritmetika')
2 print(21+22) # 43
3 print(34-14) # 20
4 print(2*3) # 6
5 print(21/2) # 10.5
6 print(21.00/2.00) # 10.5
7 print(21%2) # 1
8 print(21.00//2.00) # 10.0
9 print(4**3) # 4 pangkat 3
10 print('Operator Relasional / Perbandingan')
11 print(3>22) # False
12 print(3<22) # True
13 print(4<=4) # True
14 print(4>=4) # True
15 print(5==5.0) # True
16 print(1!=1.0) # False
17 print('Operator Bitwise')
18 x = 25 # nilai awal
19 # 25 = 0001 1001
20 print(x >> 2) # 0000 0110 = 6
21 print(x << 2) # 0001 1000 = 24
22 a = 3 # 0000 0011
23 b = 6 # 0000 0110
24 # AND
25 print (a & b) # jika bit di dua operand sama, diaktifkan di hasil
26 # 0000 0010 = 2
27 # OR
28 print (a | b) # jika bit ada di salah satu atau kedua operand,
29 # diaktifkan di hasil:
30 # 0000 0111 = 7
31 # XOR
32 print (a ^ b) # jika bit ada di salah satu operand tapi tdk di keduanya
```

```
33     ,          # diaktifkan di hasil:
34     # 0000 0101 = 5
35 # Negasi / Not
36 print (-a)
37 print('Operator Penugasan / Assignment')
38 x = 50
39 print(x) # 50
40 x+=5
41 print(x) # x = x lama + 5 = 50 + 5 = 55
42 x-=2
43 print(x) # x = x lama - 2 = 55 - 2 = 53
44 x*=2
45 print(x) # x = x lama * 2 = 53 * 2 = 106
46 x/=2
47 print(x) # x = x lama / 2 = 106 / 2 = 53
48 x%=3
49 print(x) # x = x lama modulo 3 = 53 modulo 3 = 2.0
50     # (karena pembagian terakhir berhenti di 51)
51 x = 55
52 x//=2
53 print(x) # x = x lama / 2, hasil dibulatkan ke bawah = 27.5
54     # dibulatkan 27
55 x**=2
56 print(x) # x = x lama pangkat 2 = 27 pangkat 2 = 729
57 x = 7
58 x&=2
59 print(x) # x = x lama AND 2 = 7 and 2
60     # 7 = 0000 0111
61     # 2 = 0000 0010
62     # bit hidup jika di kedua operand hidup
63     # 0000 0010 = 2
64 x = 7
65 x|=2
66 print(x) # x = x lama OR 2 = 7 or 2
67     # 7 = 0000 0111
68     # 2 = 0000 0010
69     # bit hidup jika di salah satu operand hidup
70     # 0000 0111 = 7
71 x = 7
72 x^=2
73 print(x) # x = x lama XOR 2 = 7 xor 2
74     # 7 = 0000 0111
```

```
75         # 2 = 0000 0010
76         # bit hidup jika di salah satu operand hidup,
77         # tapi tidak di keduanya
78         # 0000 0101 = 5
79 x = 7
80 x>>=2
81 print(x) # x = x lama >> 2 = 7 >> 2
82         # 7 = 0000 0111
83         # 0000 0001 = 1
84 x = 7
85 x<<=2
86 print(x) # x = x lama << 2 = 7 << 2
87         # 7 = 0000 0111
88         # 0001 1100 = 28
89 print('Operator Logika')
90 x = 3 > 1 and 2 < 19 # jika kedua sisi true -> true
91 print(x)
92 x = 3 > 4 or 2 < 10 # jika salah satu sisi benar -> true
93 print(x)
94 x = not(3 > 4) # not -> negasi
95 print(x)
96 print('Operator Keanggotaan / Membership')
97 x = (2,5,9,8,1,9,7,2)
98 print(9 in x)
99 print(10 in x)
100 print(10 not in x)
101 print('Operator Identitas / Identity')
102 x = 7
103 print(x is 7)
104 print(x is not 7)
```

Indentasi

Source code Python mewajibkan adanya indentasi untuk pengelompokan. Sebagai contoh:

```
1 a = (2,5,8,1,9,7,2)
2 for x in a:
3     print(x) # harus ditulis dalam indentasi karena merupakan bagian
              kelompok
4     # dari for x
```


Secara umum, biasanya digunakan spasi (bukan tab) sebanyak 4 karakter.

Ekspresi

Ekspresi merupakan gabungan dari nilai, variabel, pemanggilan fungsi (untuk melakukan suatu hal tertentu) yang akan dievaluasi. Setiap baris dalam source code Python biasanya berisi ekspresi. Ekspresi ini akan dievaluasi oleh interpreter Python (istilah umum: dieksekusi / dijalankan). Contoh pada baris-baris pembahasan tentang operator di atas merupakan ekspresi.

Pengendalian Alur Program

if

Python menyediakan if ... elif ... else serta variasinya untuk keperluan jika terjadi kondisi tertentu dalam aliran program dan diteruskan dengan suatu ekspresi tertentu. Bentuk dari pernyataan if ini adalah sebagai berikut:

```
1 nilai = int(input("Masukkan nilai siswa = "))
2
3 if nilai > 60:
4     print("Lulus")
5 else:
6     print("Tidak lulus")
7
8 if nilai <= 10:
9     print("Anda harus mengulang semester depan")
10
11 ipsemester = float(input("Masukkan nilai IP semester = "))
12
13 if ipsemester > 3:
14     print("Anda bisa mengambil 24 SKS")
15 elif ipsemester >= 2.75:
16     print("Anda bisa mengambil 21 SKS")
17 elif ipsemester >= 2:
18     print("Anda bisa mengambil 18 SKS")
19 else:
20     print("Anda hanya bisa mengambil 12 SKS")
```

while

1 Pernyataan **while** digunakan untuk menjalankan perintah ataupun ekspresi di dalam blok **while** selama kondisi masih bernilai True.

```
1 nilai = 1
2
3 # akan ditampilkan angka 1 - 9
4 # setelah itu berhenti
5 while nilai < 10:
6     print(nilai)
7     nilai += 1
8
9 while nilai <= 20:
10    print(nilai)
11    nilai += 1
12 else:
13    print("nilai sudah mencapai 20 ...")
14
15 input("Tekan sembarang tombol untuk meneruskan ... ")
16
17 # akan ditampilkan angka 21
18 # dan seterusnya tidak akan berhenti
19 # kecuali ditekan Ctrl-C
20 while True:
21    print(nilai)
22    nilai += 1
```

for

1 Pernyataan **for** digunakan untuk melakukan iterasi sepanjang list / daftar maupun string.

```
1 daftar = ["first", "2nd", 3]
2
3 for a in daftar:
4     print(a)
5
6 for a in range(20):
7     print(a) # 0 - 19
8
9 for a in range(1, 5):
10    print(a) # 1 -4
11
12 for w in 'ABCDEFGH':
```

```
13     print(w)
14
15 # range(start, stop, step)
16 for a in range(1, 5, 2):
17     print(a) # 1, 3
18 else:
19     print("bukan selisih 2")
20
21 for a in range(20):
22     if a > 0 and a % 2 == 0:
23         print(a, " habis dibagi dua")
24     else:
25         print(a, " ganjil")
```

Fungsi

Apakah Fungsi Itu?

Fungsi (function) merupakan blok / unit dari program yang digunakan untuk mengerjakan suatu pekerjaan tertentu dan mengembalikan hasilnya ke pemanggil. Fungsi dimaksudkan utk membuat reusable code. Seringkali dalam memprogram, seorang pemrogram mengerjakan hal-hal yang kadang sama dan diulang berkali-kali. Untuk membuat supaya tidak perlu mengerjakan tugas yang berulang-ulang tersebut, kode program dimasukkan dalam fungsi. saat diperlukan, fungsi tersebut dipanggil.

Membuat Fungsi

Rerangka fungsi di Python adalah sebagai berikut:

```
1 def nama(arg1, arg2, ... , argN):
2     ...
3     Isi fungsi
4     ...
5
6 # memanggil fungsi:
7
8 retvalnama = nama(arg1, arg2, ... , argN)
```

Contoh:

```
1 # function.py
2 def jumlah(arg1):
3     jml = 0
4     for a in arg1:
5         jml += 1
6     return jml
7
8 str_obj = "Wabi Teknologi Indonesia"
```

```
9  jml_str = jumlah(str_obj)
10
11  print(f'String {str_obj} mempunyai ' + str(jml_str) + ' karakter')
```

Fungsi Dengan Argumen Default

Suatu fungsi bisa mempunyai argumen default dengan cara menetapkan nilainya secara langsung pada definisi fungsi. Jika saat pemanggilan fungsi tersebut tidak menyertakan argumen, maka argumen default tersebut yang digunakan.

```
1  # function_default.py
2  #
3  # menghitung jumlah karakter tertentu dalam string
4  # jika the_char tidak ada, maka default menghitung
5  # jumlah semua karakter
6
7  def jumlah(the_str, the_char=None):
8      jml = 0
9      if the_char:
10         for a in the_str:
11             if a == the_char:
12                 jml += 1
13     else:
14         for a in the_str:
15             jml += 1
16     return jml
17
18  str_obj = "Wabi Teknologi Indonesia"
19
20  jml_semua = jumlah(str_obj)
21  print(f'String {str_obj} mempunyai ' + str(jml_semua) + ' karakter')
22
23  jml_a = jumlah(str_obj, 'a')
24  print(f'String {str_obj} mempunyai ' + str(jml_a) + ' karakter a')
```

Fungsi Dengan Argumen Tidak Pasti

Jika jumlah argumen tidak pasti, pemrogram bisa menggunakan `*args` (untuk argumen tanpa key) dan `**kwargs` (untuk argumen dengan key dan value).

```
1 # function_args_kwargs.py
2
3 def penambah(*args):
4     total = 0
5     for op in args:
6         total += op
7     return total
8
9 jml1 = penambah(1)
10 jml2 = penambah(23, 24, 21)
11 jml3 = penambah(1,2,3,4,5,6,7,8,9,10)
12
13 print(jml1)
14 print(jml2)
15 print(jml3)
16
17 def gabungkan(*args, pemisah='/'):
18     return pemisah.join(args)
19
20 str_gabung = gabungkan('a','b','c')
21 str_gabung2 = gabungkan('a','b','c', pemisah='-')
22
23 print(str_gabung)
24 print(str_gabung2)
25
26 def get_kwargs(**kwargs):
27     return kwargs
28
29 def get_key_value(**kwargs):
30     for key, value in kwargs.items():
31         print("Nilai {} = {}".format(key, value))
32
33 kw1 = get_kwargs(product_id='P001', product_name='Laptop')
34 print(kw1)
35 get_key_value(product_id='P001', product_name='Laptop')
```

Ekspresi / Operator / Fungsi Lambda

1 Fungsi lambda merupakan fungsi kecil dan tanpa nama. Penggunaannya mirip dengan fungsi biasa, tetapi meski bisa menggunakan banyak

argumen, fungsi lambda hanya terdiri atas 1 ekspresi yang dieksekusi dan langsung dikembalikan nilainya ke pemanggilnya. Penggunaan dari fungsi lambda ini antara lain untuk:

- Higher order function (lihat materi Functional Programming) atau fungsi yang mempunyai argumen berupa fungsi,
- Digunakan bersama struktur data di Python (misalnya untuk map dan filter di list).
- Digunakan untuk membuat fungsi secara cepat dan dalam waktu pendek saja.

```
1 # lambda1.py
2
3 # definisi:
4 # lambda arg1, arg2, ... , argN: ekspresi
5
6 multiple = lambda x, y: x * y
7
8 print(multiple(10,20))
9
10 def kali_berapa(n):
11     return lambda a: a * n
12
13 kali_dua = kali_berapa(2)
14
15 print(kali_dua(90))
```


Struktur Data di Python

Struktur data merupakan pengorganisasian, pengelolaan, serta penyimpanan data untuk akses data yang efisien. Python mempunyai beberapa model data.

List

List digunakan untuk menyimpan data (biasanya) dari tipe yang sama (meski tidak selalu harus sama) dalam suatu rangkaian data yang dipisahkan oleh koma dalam kurung kotak.

```
1 # list.py
2
3 daftar_makanan = ['soto', 'bakso', 'pece1', 'nila bakar']
4
5 print(daftar_makanan)
6 # hasil: ['soto', 'bakso', 'pece1', 'nila bakar']
7
8 print()
9 for makanan in daftar_makanan:
10     print(makanan)
11     # hasil:
12     #   soto
13     #   bakso
14     #   pece1
15     #   nila bakar
16
17 print()
18 print(daftar_makanan[0])
19 # soto
20
21 print()
22 print(daftar_makanan[-1])
23 # nila bakar
24
25 print()
```

```
26 print(daftar_makanan[-3])
27 # bakso
28
29 print()
30 print(daftar_makanan[-2:])
31 # hasil: ['pecel', 'nila bakar']
32
33 print()
34 print(daftar_makanan[:])
35 # hasil: ['soto', 'bakso', 'pecel', 'nila bakar']
36
37 daftar2 = daftar_makanan + ['oseng tempe', 'sayur pisang']
38 print()
39 print(daftar2)
40 # hasil: ['soto', 'bakso', 'pecel', 'nila bakar', 'oseng tempe', 'sayur
    pisang']
41 jml_makanan = len(daftar2)
42 print(f'ada {jml_makanan} jumlah makanan')
43
44 # list bersifat mutable
45 daftar2[1] = 'mie ayam'
46 print()
47 print(daftar2)
48 # hasil: ['soto', 'mie ayam', 'pecel', 'nila bakar', 'oseng tempe', '
    sayur pisang']
49
50 # index 2 diganti sampai sebelum index 4
51 daftar2[2:4] = ['pecel lele', 'nila goreng']
52 print()
53 print(daftar2)
54 # hasil: ['soto', 'mie ayam', 'pecel lele', 'nila goreng', 'oseng tempe
    ', 'sayur pisang']
55
56 # list bisa berada di dalam list
57 daftar2[1] = ['mie ayam biasa', 'mie ayam bakso', 'mie ayam istimewa']
58 print()
59 print(daftar2)
60 # hasil: ['soto', ['mie ayam biasa', 'mie ayam bakso', 'mie ayam
    istimewa'],
61 #         'pecel lele', 'nila goreng', 'oseng tempe', 'sayur pisang']
62 print(daftar2[1])
63 # hasil: ['mie ayam biasa', 'mie ayam bakso', 'mie ayam istimewa']
64 print(daftar2[1][2])
```

```
65 # hasil: mie ayam istimewa
```

Python menyediakan banyak fungsi untuk memanipulasi list, silahkan melihat selengkapnya dengan perintah `help(list)` dari prompt Python

Tuple

Tuple mirip dengan list, tetapi beda kurung dan bersifat immutable (tidak bisa diubah).

```
1 # tuple.py
2
3 the_data = 234, 'data 1', 'data 2', 343
4 print(the_data)
5 # hasil: (234, 'data 1', 'data 2', 343)
6
7 print(the_data[2])
8 # hasil: data2
9
10 # the_data[2] = 'change this'
11 # error: TypeError: 'tuple' object does not support item assignment
12
13 data2 = 'data x', 'data y', (123, 321)
14 print(data2)
15 # hasil: ('data x', 'data y', (123, 321))
16 print(data2[2][1])
17 # hasil: 321
18 for a in data2:
19     print(a)
20 # hasil:
21 #   data x
22 #   data y
23 #   (123, 321)
24
25 # membuat tuple yang hanya berisi 1:
26 data3 = 435,
27 print(data3)
28 # hasil: (435,)
```

Lihat juga `help(tuple)`.

Sets

Sets merupakan struktur data untuk koleksi yang tidak terurut tetapi tidak membolehkan lebih dari satu nilai yang sama dalam setiap koleksi tersebut.

```
1 # set.py
2
3 proglang = {'Rust', 'Python', 'Go', 'Rust'}
4 print(proglang)
5 tambahan = ('Ruby', 'Lua')
6 proglang.add(tambahan)
7 print(proglang)
8 print('Rust' in proglang)
9
10 huruf = set('Wabi Teknologi')
11 print(huruf)
12
13 huruf2 = set()
14 print(huruf2)
15 huruf2.add('Wabi Teknologi')
16 print(huruf2)
17
18 kata1 = set('indonesia')
19 kata2 = set('merdeka')
20 print(kata1)
21 print(kata2)
22
23 # ada di kata1, tidak ada di kata2
24 print(kata1 - kata2)
25
26 # ada di kata1 atau di kata2 atau di keduanya
27 print(kata1 | kata2)
28
29 # ada di kata1 dan kata2
30 print(kata1 & kata2)
31
32 # ada di kata1 atau di kata2 tapi tidak di keduanya
33 print(kata1 ^ kata2)
```

Lihat juga `help(set)`.

Dictionary

1 Struktur data ini mengorganisasikan data dalam bentuk seperti kamus: ada key dan value untuk key tersebut.

```
1 # dict.py
2
3 rumah = {'H-304': 'Bambang Purnomosidi', 'H-303': 'Anton'}
4 print(rumah)
5 print(rumah.items())
6 print(rumah['H-304'])
7 for k, v in rumah.items():
8     print(f'Rumah nomor {k} adalah tempat tinggal keluarga {v}')
9
10 print('H-304' in rumah)
11 print('H-305' in rumah)
12 print('H-304' not in rumah)
13 print(sorted(rumah))
```

Lihat juga `help(dict)`

Modul dan Conda

Modul Standar

Python menyediakan berbagai macam fungsi dan modul standar yang bisa dipakai langsung tanpa perlu menggunakan pustaka pihak ketiga. Modul standar selengkapnya bisa dilihat di <https://docs.python.org/3/library/index.html>.

Modul yang Didefinisikan Pemakai (User Defined Module)

Kumpulan fungsi (dan nantinya class) yang sudah dibuat bisa disimpan dalam suatu file dan digunakan sebagai modul. Modul sering juga disebut sebagai paket (package). Modul ini berguna untuk reusable code.

```
1 # modul-01.py
2
3 def penambah(*args):
4     total = 0
5     for op in args:
6         total += op
7     return total
8
9 # jika di-import, maka __name__ berisi nama modul yaitu
10 # namafile.py
11 # jika dijalankan dari shell / command line, maka
12 # __name__ akan berisi '__main__'
13 # jadi, di bawah ini tidak akan dijalankan jika di-import
14 if __name__ == '__main__':
15     print(penambah(32,43,12))
```

Jika dipanggil dari command line / shell:

```
1 » python modul01.py
2 87
```

Jika di-import:

```
1 # pthon
2 » python
3 Python 3.7.1 (default, Oct 22 2018, 10:41:28)
4 [GCC 8.2.1 20180831] on linux
5 Type "help", "copyright", "credits" or "license" for more information.
6 >>> import modul01
7 >>> modul01.penambah(12,23,12,32)
8 79
9 >>>
```

Saat menemui perintah import modul01, python akan mencari isi dari modul standar. Jika tidak ada, maka akan dicari modul01.py pada:

- Direktori aktif saat itu
- Isi dari \$PYTHONPATH
- Isi dari sys.path:

```
1 >>> import sys
2 >>> sys.path
3 ['', '/usr/lib/python37.zip', '/usr/lib/python3.7', '/usr/lib/python3
  .7/lib-dynload', '/usr/lib/python3.7/site-packages']
4 >>>
```

pip

Python menyediakan perintah untuk mengelola pustaka pihak ketiga jika pemrogram ingin menggunakan berbagai pustaka tersebut untuk keperluan tugas pemrograman yang diberikan ke pemrogram. Secara default, perintah yang digunakan adalah pip.

```
1 » pip list
2 Package                                Version
3 -----
4 alabaster                              0.7.11
5 anytree                                2.4.3
6 appdirs                                1.4.3
7 Babel                                   2.6.0
8 backcall                                0.1.0
9 Brlapi                                  0.6.7
10 btrfsutil                              1.0.0
```

```
11 CacheControl          0.12.5
12 chardet                3.0.4
13 colorama               0.4.1
14 decorator              4.3.0
15 distlib                0.2.8
16 distro                  1.3.0
17 docutils                0.14
18 greenlet                0.4.15
19 html5lib                1.0.1
20 ...
21 ...
```

```
1 Untuk menginstall paket, gunakan:
```

```
1 » pip install jupyter
2 Collecting jupyter
3   Using cached https://files.pythonhosted.org/packages/83/df/0
      f5dd132200728a86190397e1ea87cd76244e42d39ec5e88efd25b2abd7e/
      jupyter-1.0.0-py2.py3-none-any.whl
4 Collecting ipywidgets (from jupyter)
5 ...
6 ...
```

pip masih mempunyai banyak opsi, silahkan melihat menggunakan perintah `pip -help`.

Conda

Conda merupakan pengelola paket dan lingkungan Python yang dibuat oleh Anaconda, Inc. Hampir mirip dengan pip, hanya saja paket yang ada merupakan paket yang sudah diaudit dan dikelola dengan baik oleh Anaconda, Inc. Untuk mengelola paket, berikut adalah beberapa perintah dasar dari conda:

```
1 conda list => daftar paket yang sudah terinstall
2 conda install x => install paket x
3 conda remove x => uninstall paket x
4 conda update x => update paket x
5 conda update --all => update semua paket
```

Selain mengelola paket, conda juga bisa digunakan untuk mengelola lingkungan karena seringkali pemrogram memerlukan python versi tertentu yang berbeda dengan yang telah diinstall (baik di level

sistem operasi maupun di conda/anaconda). Berikut ini adalah beberapa perintah yang bisa digunakan untuk mengelola environment:

- `conda env list` => melihat env apa saja yang ada
- `conda activate nama-env` => mengaktifkan environment
- `conda create -p /home/bpdp/py36 python=3.6` => membuat environment dengan versi Python tertentu

Operasi I/O

Input dari Keyboard

Untuk menerima input dari keyboard, digunakan input

```
1 # input_keyboard.py
2
3 nama = input('Masukkan nama = ')
4 usia = int(input('Umur = '))
5
6 print(f>Nama = {nama}, usia = {usia}')
7 print('Nama = {0:^}, usia = {1:4d}'.format(nama, usia))
```

Output ke Layar

Untuk menampilkan output, digunakan f di awal string atau menggunakan format seperti pada contoh di atas.

Mengambil Isi File

Untuk mengambil isi file, buka menggunakan open kemudian gunakan read untuk membaca isi. Mode pada open disesuaikan dengan tujuan pembukaan file. Menulis ke file dilakukan dengan write().

```
1 # open_file_with.py
2
3 # default: tanpa argumen mode, dibuka utk dibaca (read)
4 with open('angka.txt') as f:
5     read_data = f.read()
6     print(read_data)
7     # error: io.UnsupportedOperation: not writable
```

```
8     #f.write('tambahan 1')
9
10  # dengan 'with', tidak perlu di close
11  print(f.closed)
12
13  # r+ => read write
14  with open('angka.txt','r+') as f:
15      read_data = f.read()
16      print(read_data)
17      # bisa ditulisi karena r+
18      f.write('tambahan')
19
20  # r+ => read write
21  # dibuka dengan w+ membuat isi hilang
22  with open('angka.txt','w') as f:
23      f.write('tambahan')
24
25  with open('angka.txt') as f:
26      read_data = f.read()
27      print(f'sekarang hanya berisi 1 baris: {read_data}')
28
29  # sekarang diisi dengan loop angka
30  with open('angka.txt', 'w') as f:
31      for a in range(1,11):
32          f.write(str(a) + '\n')
33
34  # tampilkan hasil pengisian
35  with open('angka.txt') as f:
36      for line in f:
37          print(line, end='')

```

Menangani Error dan Exception

Saat membuat program, seorang pemrogram tidak akan terlepas dari kondisi-kondisi yang terkait dengan program yang dia buat, khususnya kemungkinan terjadinya kesalahan. Python mempunyai berbagai macam konstruksi untuk mendeteksi error (yang paling sederhana, misalnya `SyntaxError`) jika terdapat hal-hal yang bisa diketahui kesalahannya sejak awal. Meski demikian, sering kali jika tidak ada error juga tidak menjamin bahwa saat dijalankan tidak akan terjadi hal-hal yang di luar dugaan. Tugas pemrogram adalah mengantisipasi berbagai macam kondisi tersebut.

Saat terjadi error, pemrogram melihat pada error yang terjadi, setelah itu memperbaiki berdasarkan error yang muncul. Seringkali hal ini juga melibatkan pembacaan manual / dokumentasi dan penggunaan sumber daya di Internet (StackOverflow dan lain-lain) untuk melihat kemungkinan solusi. Berikut ini adalah contoh perintah-perintah yang menimbulkan error (kata yang dicetak tebal tambahan dari penulis untuk menjelaskan nilai error):

```
1 >>> f = open('abc.txt')
2 Traceback (most recent call last):
3   File "<stdin>", line 1, in <module>
4   FileNotFoundError: [Errno 2] No such file or directory: 'abc.txt'
5 >>> f = open('abca.txt','f')
6 Traceback (most recent call last):
7   File "<stdin>", line 1, in <module>
8   ValueError: invalid mode: 'f'
9 >>> print 'abcdefg'
10    File "<stdin>", line 1
11        print 'abcdefg'
12                ^
13 SyntaxError: Missing parentheses in call to 'print'. Did you mean print
    ('abcdefg')?
```

Setiap terjadi error, Python akan memunculkan exception atau suatu kondisi “pengecualian”. Pemrogram biasanya harus waspada terhadap berbagai kemungkinan error serta exception yang terjadi untuk diantisipasi. Untuk mengantisipasi exception, gunakan blok `try ... except ... else ... finally`.

```
1 # except.py
2
```

```
3 import sys
4
5 # tanpa exception handling
6 # b = float(input('masukkan angka float: '))
7 # amasukkan angka float: f
8 # Traceback (most recent call last):
9 #   File "except.py", line 5, in <module>
10 #     b = float(input('masukkan angka float: '))
11 # ValueError: could not convert string to float: 'f'
12
13 # Setelah dihandle:
14 try:
15     a = float(input("masukkan angka float: "))
16 except ValueError:
17     print('harus memasukkan nilai float')
18
19 # Jika tidak tau kemungkinan error:
20 try:
21     a = float(input("masukkan angka: "))
22     b = float(input("masukkan angka pembagi: "))
23     z = a/b
24 except:
25     print("Error:", sys.exc_info()[0])
26 else:
27     print('Hasil bagi: ', z)
28 finally:
29     # bagian ini biasanya untuk clean-up, di dunia nyata
30     # biasanya berisi bagian utk close connection, menutup
31     # file dan lain-lain
32     print('Selesai')
33 # hasil:
34 # masukkan angka: 40
35 # masukkan angka pembagi: 0
36 # Error: <class 'ZeroDivisionError'>
```

OOP di Python

Python merupakan bahasa yang multiparadigm, artinya mendukung berbagai paradigma pemrograman. Dua paradigma yang akan dibahas khusus disini adalah OOP (Object-Oriented Programming) dan functional programming.

OOP merupakan paradigma pemrograman yang meniru cara pandang natural manusia dalam menyelesaikan masalah. Dalam dunia nyata, banyak terdapat obyek dan antar obyek tersebut bisa saling mengirim pesan. Dengan pesan tersebut, kolaborasi dilakukan sehingga masalah terselesaikan. Masing-masing obyek tersebut mempunyai perilaku dan karakteristik (misal dosen maupun mahasiswa mempunyai perilaku dan karakteristik masing-masing). Setiap obyek juga mempunyai kelas yang mendefinisikan perilaku dan karakteristik tersebut. Seringkali suatu kelas merupakan turunan dari kelas lain (misal dosen merupakan turunan dari manusia) dan seterusnya.

Mengikuti pola natural seperti itu, OOP menghendaki adanya definisi kelas serta pembuatan instance / obyek dari kelas tersebut. Jika belum ada yang mirip, maka bisa dibuat kelas dari awal, jika sudah ada yang mirip, maka tinggal dibuat turunannya.

```
1 # kelas.py
2
3 # definisi kelas paling sederhana
4 # bisa ditambah properties
5 class Dosen:
6     pass
7
8 bdpd = Dosen()
9 bdpd.nama = 'Bambang Purnomosidi'
10
11 print(bdpd)
12 print(bdpd.nama)
13
14 class DosenSTMIKAkom(Dosen):
15
16     institusi = 'STMIK AKAKOM'
17
18     # konstruktor
```

```
19     def __init__(self, npp, nama):
20         self.npp = npp
21         self.nama = nama
22
23     def mengajar(self, *args):
24         self.mk_diampu = args
25
26 bambangpdp = DosenSTMIKAkakom('123', 'bpdp')
27 print(bambangpdp)
28 bambangpdp.mengajar('Teknologi Cloud Computing', 'Big Data Analytics')
29 print(bambangpdp.mk_diampu)
30 print(bambangpdp.institusi)
31
32 class DosenSTMIKAkakomTI(DosenSTMIKAkakom):
33
34     prodi = 'Teknik Informatika'
35
36 nia = DosenSTMIKAkakomTI('213', 'Nia R')
37 print(nia.institusi)
38 print(nia.prodi)
```

Functional Programming di Python

Functional Programming (FP) adalah paradigma pemrograman yang memandang komputasi sebagai evaluasi dari fungsi matematis serta menghindari mutable data dan perubahan state. FP biasanya ditandai oleh berbagai fitur yang akan dibicarakan disini.

Pure Function

Suatu fungsi yang pure ditandai dengan adanya pemrosesan di badan fungsi yang sama sekali tidak terpengaruh oleh state serta variabel dari luar badan fungsi. Selain itu, definisi fungsi juga tidak menghasilkan side effects, artinya tidak menghasilkan operasi I/O yang kemungkinan bisa mengambil data dari luar maupun menghasilkan sesuatu yang bisa menjadi bottlenecks (misal koneksi ke Internet, jaringan, mengakses file, dan lain-lain).

```
1 # non_pure_function.py
2
3 a = 200
4
5 def change_state():
6
7     global a
8
9     a += 100
10
11     return a
12
13 print(change_state())
14 print(change_state())
15 print(change_state())
16 print(change_state())
17 print(change_state())
```

Untuk pure function, silahkan lihat contoh berikut:

```
1 # pure_function.py
```



```
2
3 a = 200
4
5 def no_change_state():
6
7     # jangan mengakses variable dari luar scope def func ini
8     #
9
10    return 10*10
11
12 print(no_change_state())
13 print(no_change_state())
14 print(no_change_state())
15 print(no_change_state())
16 print(no_change_state())
```

Iterator

Iterator merupakan obyek yang digunakan untuk menampung data stream. Iterator mempunyai semacam pointer untuk menyimpan posisi penyimpanan data dan bisa bergerak pada keseluruhan data tersebut untuk mengakses elemen data dalam suatu perulangan. Fungsi yang digunakan adalah iter().

```
1 # iterator.py
2
3 daftar = [1,2,3,4,5,6]
4
5 # cara iterator
6 i_daftar = iter(daftar)
7
8 print(i_daftar)
9
10 a = 1
11
12 while a < len(daftar):
13     print(next(i_daftar))
14     a += 1
15
16 # cara mudah
17 for z in daftar:
18     print(z)
```

Generator

Generator merupakan konstruksi di Python yang digunakan untuk menghasilkan iterator. Perintah yang digunakan adalah `yield`.

```
1 # generator.py
2
3 def generate_val(N):
4     for i in range(N):
5         yield i
6
7 hasil = generate_val(10)
8 print(hasil)
9
10 for a in hasil:
11     print(a)
```

Map

Map digunakan untuk melakukan sesuatu fungsi terhadap obyek yang bersifat iterable. Semua obyek sequence (seperti list) bersifat iterable, demikian juga dengan hasil dari iterator dan generator.

```
1 # map.py
2
3 def make_ucase(the_str):
4     return the_str.upper()
5
6 # a => iterable
7 a = ['a', 'b', 'c']
8
9 # kerjakan fungsi make_ucase utk setiap item a
10 b = map(make_ucase, a)
11
12 for c in b:
13     print(c)
```

Reduce

Reduce digunakan untuk mengubah obyek iterable menjadi satu nilai saja.

```
1 # reduce.py
2
3 from functools import reduce
4
5 # tanpa lambda expression dan reduce
6 hasil = 1
7 x = [1, 2, 3, 4, 5]
8 for num in x:
9     hasil = hasil * num
10
11 print(hasil)
12
13 # dengan lambda expression dan reduce
14 hasil2 = reduce((lambda x, y: x * y), [1, 2, 3, 4, 5])
15
16 print(hasil2)
17
18 # hasil:
19 # 120
20 # 120
```

Filter

Filter digunakan untuk mengambil nilai di obyek iterable dan melakukan filtering terhadap nilai tersebut akan sesuai dengan yang dikehendaki pada parameter fungsi.

```
1 # filter.py
2
3 nilai = range(-10, 10)
4
5 for a in nilai:
6     print(a)
7     # hasilL -10 sampai 10
8
9 # Kita akan memfilter list sehingga hanya yang berisi nilai positif
10 # yang akan masuk ke list baru
11
```

```
12 l_baru = list(filter(lambda angka: angka > 0, nilai))
13 print(l_baru)
14 # hasil: [1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Higher Order Function

HOF memungkinkan fungsi menjadi argumen dari suatu fungsi lain. Selain itu, dimungkinkan juga untuk membuat fungsi sebagai suatu return value.

```
1 # hof.py (higher order function)
2
3 # HOF - fungsi sebagai argumen fungsi
4 def penjumlahan(angka):
5     return sum(angka)
6
7 def aksi(func, angka2):
8     return func(angka2)
9
10 print(aksi(penjumlahan, [1, 2, 3, 4, 5]))
11
12 # HOF - fungsi sebagai return value
13 def remaja():
14     return "remaja"
15 def dewasa():
16     return "dewasa"
17
18 def person():
19     umur = int(input("Umur anda: "))
20
21     if umur <= 21:
22         return remaja()
23     else:
24         return dewasa()
25
26 print(person())
```

Closure

Closure sering juga disebut sebagai partial application, memungkinkan untuk memanggil fungsi tanpa menyediakan seluruh argumen yang dipersyaratkan.

```
1 # closure.py
2
3 from functools import partial
4
5 # bilangan pangkat eksponen
6 def pangkat(bilangan, eksponen):
7     return bilangan ** eksponen
8
9 kuadrat = partial(pangkat, eksponen=2)
10 print(kuadrat(2))
11 # hasil = 2
12
13 # parsial:
14 # pangkat dipanggil dengan arg eksponen ditetapkan di awal
15 pangkat_empat = partial(pangkat, eksponen=4)
16 print(pangkat_empat(2))
17 # hasil = 16
```

Asynchronous I/O / Concurrent Programming di Python

Concurrent programming adalah bentuk komputasi yang memungkinkan lebih dari satu tugas komputasi dikerjakan secara bersamaan, tidak dalam bentuk berurutan (sekuensial). Model komputasi ini sering disebut juga sebagai async karena suatu tugas komputasi tidak perlu menunggu tugas komputasi lainnya untuk selesai tetapi langsung menjalankan bagian komputasinya meski aliran pemrograman tetap memerlukan bagian lain tersebut. Bagian lain tetap dikerjakan sambil ditunggu bagian tersebut selesai. Setelah selesai, hasilnya baru akan diproses ke semua bagian yang menunggu hasil tersebut.

Versi Python yang diperlukan untuk concurrent programming ini adalah versi 3.7+. Ada banyak hal yang disediakan oleh Python untuk keperluan ini, tetapi disini akan dibahas tentang coroutines dan tasks.

```
1 # asynchronous.py
2
3 # diambil dari manual Python
4 # https://docs.python.org/3/library/asyncio-task.html
5
6 import asyncio
7
8 async def factorial(name, number):
9     f = 1
10    for i in range(2, number + 1):
11        print(f"Task {name}: Compute factorial({i})...")
12        await asyncio.sleep(1)
13        f *= i
14    print(f"Task {name}: factorial({number}) = {f}")
15
16 async def main():
17     # Schedule three calls *concurrently*:
18     await asyncio.gather(
19         factorial("A", 8),
20         factorial("B", 3),
```

```
21         factorial("C", 4),  
22     )  
23  
24     asyncio.run(main())
```