Pemrograman Python

Handbook

Dr. Bambang Purnomosidi D. P.

MAGISTER TEKNOLOGI INFORMASI
STMIK AKAKOM YOGYAKARTA

Daftar Isi

Tentang Buku Ini

Repo ini berisi kode sumber serta buku **Pemrograman Python** dibuat oleh Magister Teknologi Informasi STMIK Akakom. Lisensi semua dokumen pada repo ini adalah Creative Commons Atribution-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 Tnformasi - 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:

```
Magister Teknologi Informasi
STMIK Akakom
Jl. Raya Janti Karangjambe no 143
Yogyakarta 55198
Indonesia
Phone: 0274 486664
https://pascasarjana.akakom.ac.id
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 dikompillasi (*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 benyak 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, wxpthon, PyQT, PyGTK, dan lain-lain) maupun TUI (python-ncurses, dan lain-lain)
6. Bahasa pemrograman untuk membuat tools di *softwware 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 perusahan 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 iyang dimplementasikan 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** (turuan 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 meruapakan 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 algortima.

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

```
Algorithm LargestNumber
  Input: A list of numbers L.
  Output: The largest number in the list L.
  if L.size = 0 return null
  largest ← L[0]
  for each item in L, do
    if item > largest, then
      largest ← item
  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 diimplemtasikan dalam bentuk *function*.

3. **Object-oriented**: implementasi algoritma dalam bentuk langkah-langkah dengan menggnakan 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 paradigman 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

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

2. Rancang algoritma

```
0. mulai
1. meminta masukan angka pertama
2. meminta masukan angka kedua
3. pemeriksaan dan penentuan hasil:

Jika angka pertama lebih besar daripada angka kedua:
  cetak angka pertama sebagai angka terbesar
Selain itu, jika angka pertama lebih kecil daripada angka kedua
  cetak angka kedua sebagai angka terbesar
Selain itu, jika angka pertama sama dengan angka kedua
  cetak pemberitahuan bahwa tidak ada angka terbesar karena sama
Selain itu,
  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. Dokuemntasi 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, implentasikan 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
```

```
$ python max.py
Masukkan angka pertama: 5
Masukkan angka kedua: 5
Kedua angka sama, tidak ada yang terbesar
$
```

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 bsar, 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 fasiltas 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, tetepi 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:

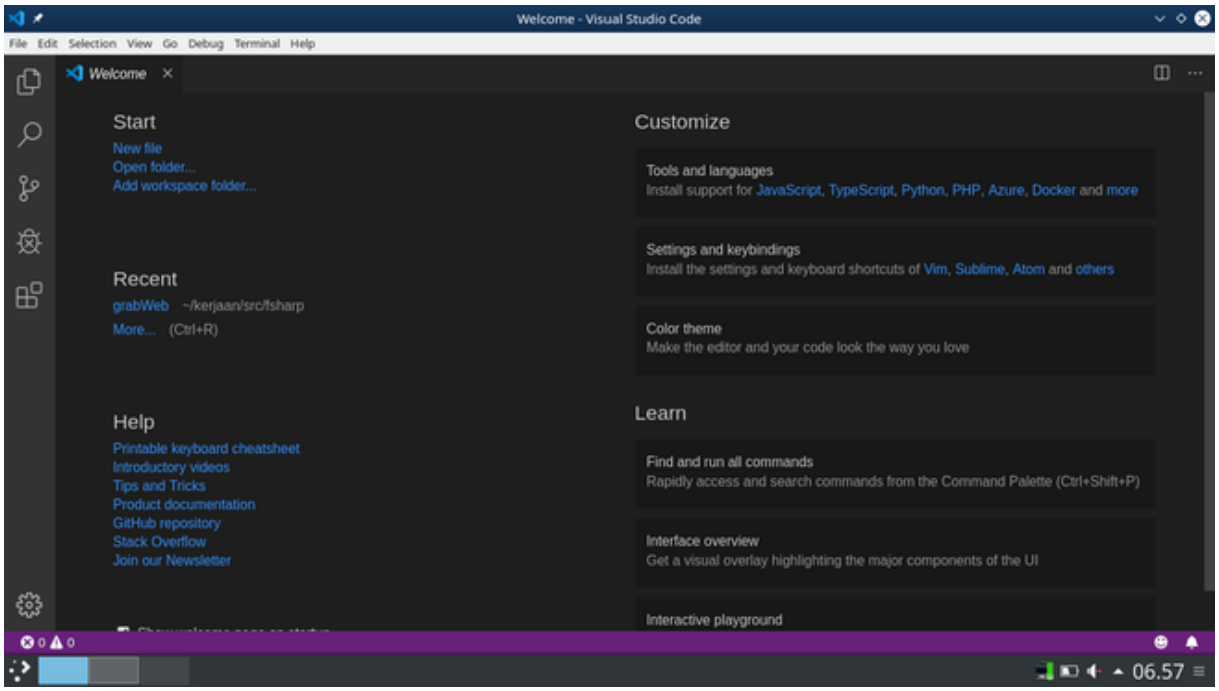

Figure 0.1: VSCode

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?itemN
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:

```
» chmod +x Miniconda3-latest-Linux-x86_64.sh
» ./Miniconda3-latest-Linux-x86_64.sh

Welcome to Miniconda3 4.7.10

In order to continue the installation process, please review the
   license
agreement.
Please, press ENTER to continue
>>>
==================================
Miniconda End User License Agreement
==================================

Copyright 2015, Anaconda, Inc.

All rights reserved under the 3-clause BSD License:

Redistribution and use in source and binary forms, with or without
   modification, are permitted provided that the following conditions
   are met:

  * Redistributions of source code must retain the above copyright
     notice, this list of conditions and the following disclaimer.
  * Redistributions in binary form must reproduce the above copyright
     notice, this list of conditions and the following disclaimer in
```

```
        the documentation and/or other materials provided with the
        distribution.
22    * Neither the name of Anaconda, Inc. ("Anaconda, Inc.") nor the names
         of its contributors may be used to endorse or promote products
        derived from this software without specific prior written
        permission.
23
24  THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS
       IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
      LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
      FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL ANACONDA,
       INC. BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
      EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
      PROCUREME NT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
      PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY
      OF LIABILI 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
      .
25
26  Notice of Third Party Software Licenses
27  ======================================
28
29  Miniconda contains open source software packages from third parties.
      These are available on an "as is" basis and subject to their
      individual license agreements. These licenses are available in
      Anaconda Distribution or at http://docs.anaconda.com/anaconda/pkg-
      docs. Any binary packages of these third party tools you obtain via
      Anaconda Distribution are subject to their individual licenses as
      well as the Anaconda license. Anaconda, Inc. reserves the right to
      change which third party tools are provided in Miniconda.
30
31  Cryptography Notice
32  ==================
33
34  This distribution includes cryptographic software. The country in which
       you currently reside may have restrictions on the import, posses
      sion, use, and/or re-export to another country, of encryption
      software. BEFORE using any encryption software, please check your
      country's laws, regulations and policies concerning the import,
      possession, or use, and re-export of encryption software, to see if
      this is permitted. See the Wassenaar Arrangement http://www.
      wassenaar.org/ for more information.
```

```
35
36 Anaconda, Inc. has self-classified this software as Export Commodity
       Control Number (ECCN) 5D992b, which includes mass market information
        security software using or performing cryptographic functions with
       asymmetric algorithms. No license is required for export of this so
       ftware to non-embargoed countries. In addition, the Intel(TM) Math
       Kernel Library contained in Anaconda, Inc.'s software is classified
       by Intel(TM) as ECCN 5D992b with no license required for export to
       non-embargoed countries.
37
38 The following packages are included in this distribution that relate to
        cryptography:
39
40 openssl
41    The OpenSSL Project is a collaborative effort to develop a robust,
          commercial-grade, full-featured, and Open Source toolkit
          implementing the Transport Layer Security (TLS) and Secure
          Sockets Layer (SSL) protocols as well as a full-strength general
          purpose cryptography library.
42
43 pycrypto
44    A collection of both secure hash functions (such as SHA256 and
          RIPEMD160), and various encryption algorithms (AES, DES, RSA,
          ElGamal, etc.).
45
46 pyopenssl
47    A thin Python wrapper around (a subset of) the OpenSSL library.
48
49 kerberos (krb5, non-Windows platforms)
50    A network authentication protocol designed to provide strong
          authentication for client/server applications by using secret-key
           cryptography.
51
52 cryptography
53     A Python library which exposes cryptographic recipes and primitives
          .
54
55
56 Do you accept the license terms? [yes|no]
57 [no] >>> yes
58
59 Miniconda3 will now be installed into this location:
60 /home/bpdp/miniconda3
```

```
61
62    - Press ENTER to confirm the location
63    - Press CTRL-C to abort the installation
64    - Or specify a different location below
65
66  [/home/bpdp/miniconda3] >>> /opt/software/python-dev-tools/miniconda3
67  PREFIX=/opt/software/python-dev-tools/miniconda3
68  Unpacking payload ...
69  Collecting package metadata (current_repodata.json): done
70  Solving environment: done
71
72  ## Package Plan ##
73
74    environment location: /home/bpdp/tmp/miniconda3
75
76    added / updated specs:
77      - _libgcc_mutex==0.1=main
78      - asn1crypto==0.24.0=py37_0
79      - bzip2==1.0.8=h7b6447c_0
80      - ca-certificates==2019.5.15=0
81      - certifi==2019.6.16=py37_0
82      - cffi==1.12.3=py37h2e261b9_0
83      - chardet==3.0.4=py37_1
84      - conda-package-handling==1.3.11=py37_0
85      - conda==4.7.10=py37_0
86      - cryptography==2.7=py37h1ba5d50_0
87      - idna==2.8=py37_0
88      - libarchive==3.3.3=h5d8350f_5
89      - libedit==3.1.20181209=hc058e9b_0
90      - libffi==3.2.1=hd88cf55_4
91      - libgcc-ng==9.1.0=hdf63c60_0
92      - libstdcxx-ng==9.1.0=hdf63c60_0
93      - libxml2==2.9.9=hea5a465_1
94      - lz4-c==1.8.1.2=h14c3975_0
95      - lzo==2.10=h49e0be7_2
96      - ncurses==6.1=he6710b0_1
97      - openssl==1.1.1c=h7b6447c_1
98      - pip==19.1.1=py37_0
99      - pycosat==0.6.3=py37h14c3975_0
100     - pycparser==2.19=py37_0
101     - pyopenssl==19.0.0=py37_0
102     - pysocks==1.7.0=py37_0
103     - python-libarchive-c==2.8=py37_11
```

```
    - python==3.7.3=h0371630_0
    - readline==7.0=h7b6447c_5
    - requests==2.22.0=py37_0
    - ruamel_yaml==0.15.46=py37h14c3975_0
    - setuptools==41.0.1=py37_0
    - six==1.12.0=py37_0
    - sqlite==3.29.0=h7b6447c_0
    - tk==8.6.8=hbc83047_0
    - tqdm==4.32.1=py_0
    - urllib3==1.24.2=py37_0
    - wheel==0.33.4=py37_0
    - xz==5.2.4=h14c3975_4
    - yaml==0.1.7=had09818_2
    - zlib==1.2.11=h7b6447c_3
    - zstd==1.3.7=h0b5b093_0


The following NEW packages will be INSTALLED:

  _libgcc_mutex      pkgs/main/linux-64::_libgcc_mutex-0.1-main
  asn1crypto         pkgs/main/linux-64::asn1crypto-0.24.0-py37_0
  bzip2              pkgs/main/linux-64::bzip2-1.0.8-h7b6447c_0
  ca-certificates    pkgs/main/linux-64::ca-certificates-2019.5.15-0
  certifi            pkgs/main/linux-64::certifi-2019.6.16-py37_0
  cffi               pkgs/main/linux-64::cffi-1.12.3-py37h2e261b9_0
  chardet            pkgs/main/linux-64::chardet-3.0.4-py37_1
  conda              pkgs/main/linux-64::conda-4.7.10-py37_0
  conda-package-han~ pkgs/main/linux-64::conda-package-handling-1.3.11-
     py37_0
  cryptography       pkgs/main/linux-64::cryptography-2.7-
     py37h1ba5d50_0
  idna               pkgs/main/linux-64::idna-2.8-py37_0
  libarchive         pkgs/main/linux-64::libarchive-3.3.3-h5d8350f_5
  libedit            pkgs/main/linux-64::libedit-3.1.20181209-
     hc058e9b_0
  libffi             pkgs/main/linux-64::libffi-3.2.1-hd88cf55_4
  libgcc-ng          pkgs/main/linux-64::libgcc-ng-9.1.0-hdf63c60_0
  libstdcxx-ng       pkgs/main/linux-64::libstdcxx-ng-9.1.0-hdf63c60_0
  libxml2            pkgs/main/linux-64::libxml2-2.9.9-hea5a465_1
  lz4-c              pkgs/main/linux-64::lz4-c-1.8.1.2-h14c3975_0
  lzo                pkgs/main/linux-64::lzo-2.10-h49e0be7_2
  ncurses            pkgs/main/linux-64::ncurses-6.1-he6710b0_1
  openssl            pkgs/main/linux-64::openssl-1.1.1c-h7b6447c_1
```

```
144    pip                 pkgs/main/linux-64::pip-19.1.1-py37_0
145    pycosat             pkgs/main/linux-64::pycosat-0.6.3-py37h14c3975_0
146    pycparser           pkgs/main/linux-64::pycparser-2.19-py37_0
147    pyopenssl           pkgs/main/linux-64::pyopenssl-19.0.0-py37_0
148    pysocks             pkgs/main/linux-64::pysocks-1.7.0-py37_0
149    python              pkgs/main/linux-64::python-3.7.3-h0371630_0
150    python-libarchive~ pkgs/main/linux-64::python-libarchive-c-2.8-
           py37_11
151    readline            pkgs/main/linux-64::readline-7.0-h7b6447c_5
152    requests            pkgs/main/linux-64::requests-2.22.0-py37_0
153    ruamel_yaml         pkgs/main/linux-64::ruamel_yaml-0.15.46-
           py37h14c3975_0
154    setuptools          pkgs/main/linux-64::setuptools-41.0.1-py37_0
155    six                 pkgs/main/linux-64::six-1.12.0-py37_0
156    sqlite              pkgs/main/linux-64::sqlite-3.29.0-h7b6447c_0
157    tk                  pkgs/main/linux-64::tk-8.6.8-hbc83047_0
158    tqdm                pkgs/main/noarch::tqdm-4.32.1-py_0
159    urllib3             pkgs/main/linux-64::urllib3-1.24.2-py37_0
160    wheel               pkgs/main/linux-64::wheel-0.33.4-py37_0
161    xz                  pkgs/main/linux-64::xz-5.2.4-h14c3975_4
162    yaml                pkgs/main/linux-64::yaml-0.1.7-had09818_2
163    zlib                pkgs/main/linux-64::zlib-1.2.11-h7b6447c_3
164    zstd                pkgs/main/linux-64::zstd-1.3.7-h0b5b093_0
165
166
167  Preparing transaction: done
168  Executing transaction: done
169  installation finished.
170  Do you wish the installer to initialize Miniconda3
171  by running conda init? [yes|no]
172  [no] >>> yes
173  no change     /home/bpdp/tmp/miniconda3/condabin/conda
174  no change     /home/bpdp/tmp/miniconda3/bin/conda
175  no change     /home/bpdp/tmp/miniconda3/bin/conda-env
176  no change     /home/bpdp/tmp/miniconda3/bin/activate
177  no change     /home/bpdp/tmp/miniconda3/bin/deactivate
178  no change     /home/bpdp/tmp/miniconda3/etc/profile.d/conda.sh
179  no change     /home/bpdp/tmp/miniconda3/etc/fish/conf.d/conda.fish
180  no change     /home/bpdp/tmp/miniconda3/shell/condabin/Conda.psm1
181  no change     /home/bpdp/tmp/miniconda3/shell/condabin/conda-hook.ps1
182  no change     /home/bpdp/tmp/miniconda3/lib/python3.7/site-packages/
       xontrib/conda.xsh
183  no change     /home/bpdp/tmp/miniconda3/etc/profile.d/conda.csh
```

```
184 modified      /home/bpdp/.bashrc
185
186 ==> For changes to take effect, close and re-open your current shell.
    <==
187
188 If you'd prefer that conda's base environment not be activated on
    startup,
189    set the auto_activate_base parameter to false:
190
191 conda config --set auto_activate_base false
192
193 Thank you for installing Miniconda3!
194 »
```

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 3.7.4 (default, Aug 13 2019, 20:35:49)
6  [GCC 7.3.0] :: 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
```

```
                packages.
    help        Displays a list of available conda commands and their
       help
                strings.
    info        Display information about current conda install.
    init        Initialize conda for shell interaction. [Experimental]
    install     Installs a list of packages into a specified conda
                environment.
    list        List linked packages in a conda environment.
    package     Low-level conda package utility. (EXPERIMENTAL)
    remove      Remove a list of packages from a specified conda
       environment.
    uninstall   Alias for conda remove.
    run         Run an executable in a conda environment. [
       Experimental]
    search      Search for packages and display associated information
       . The
                input is a MatchSpec, a query language for conda
                   packages.
                See examples below.
    update      Updates conda packages to the latest compatible
       version.
    upgrade     Alias for conda update.

optional arguments:
  -h, --help     Show this help message and exit.
  -V, --version  Show the conda version number and exit.

conda commands available from other packages:
  env
»
```

Jika menggunakan shell Bash:

```
» cat env-bash/anaconda/miniconda3
export PATH=/opt/software/python-dev-tools/miniconda3/bin:$PATH
» source env-bash/anaconda/miniconda3
» python
Python 3.7.4 (default, Aug 13 2019, 20:35:49)
[GCC 7.3.0] :: Anaconda, Inc. on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

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

REPL digunakan untuk keperluan mencoba potongan source code:

```
» python
Python 3.7.4 (default, Aug 13 2019, 20:35:49)
[GCC 7.3.0] :: Anaconda, Inc. on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> help
Type help() for interactive help, or help(object) for help about object
    .
>>> help()

Welcome to Python 3.7's help utility!

If this is your first time using Python, you should definitely check
    out
the tutorial on the Internet at https://docs.python.org/3.7/tutorial/.

Enter the name of any module, keyword, or topic to get help on writing
Python programs and using Python modules.  To quit this help utility
    and
return to the interpreter, just type "quit".

To get a list of available modules, keywords, symbols, or topics, type
"modules", "keywords", "symbols", or "topics".  Each module also comes
with a one-line summary of what it does; to list the modules whose name
```

```
or summary contain a given string such as "spam", type "modules spam".

help> for

help> while

help>
You are now leaving help and returning to the Python interpreter.
If you want to ask for help on a particular object directly from the
interpreter, you can type "help(object)".  Executing "help('string')"
has the same effect as typing a particular string at the help> prompt.
>>>>
```

Contoh penggunaan:

```
>>> for i in range(5):
... print(i)
...
0
1
2
3
4
5
>>> a = 20
>>> a
20
>>>
```

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 seseuatu. 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 dengan 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

```
# hello.py
print('Halo')
» python hello.py
Halo
»
```

Cara 2

```
# hello2.py
#!/usr/bin/env python

print('Halo')
» chmod +x hello2.py
» ./hello2.py
Halo
»
```

Cara 3

```
#hello3.py
#!/opt/software/python-dev-tools/miniconda3/bin/python

print('Halo')
» chmod +x hello2.py
» ./hello3.py
Halo
»
```

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:

```
>>> import keyword
>>> keyword.iskeyword('with')
True
```

```
>>> keyword.iskeyword('for')
True
>>> keyword.iskeyword('x')
False
```

Komentar

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

```
# komentar satu baris penuh
print‘(’abc) # komentar mulai kolom tertentu“””
 komentar
Lebih dari
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:

```
Nama_var = nilai

var01 = 20
var_02 = 30
nama_var = 'Satu dua tiga'

print(var01)
print(var_02)
print(nama_var)

# ini salah
var 01 = 21
```

Bentuk penugasan (pengisian variabel) lainnya:

```
>>> var1 = var2 = var3 = 4
>>> var1
4
>>> var2
```

```
4
>>> var3
4
>>> v1, v2, v3 = 'isi 1', 20, 43
>>> v1
'isi 1'
>>> v2
20
>>> v3
43
>>> v1, v2, v3 = 'isi 1', 4
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: not enough values to unpack (expected 3, got 2)
>>>
```

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:

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

Variabel juga bisa dihapus:

```
>>> a = 10
>>> a
10
>>> del a
>>> a
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'a' is not defined
>>>
```

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).

```
>>> sys.float_info
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)
>>> sys.int_info
sys.int_info(bits_per_digit=30, sizeof_digit=4)
>>> sys.maxsize
9223372036854775807
>>>
```

Bilangan kompleks:

```
x = 6
y = 4

z = complex(x,y);

print ("Bagian bilangan riil: ", z.real)
print ("Bagian imajiner dari: ", z.imag)
```

String

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

```
str1 = 'string 1'
str2 = "string 2"
str3 = """ini baris pertama
ini baris kedua
ini baris ketiga
"""
print(str1)
print(str2)
```

```
print(str3)
print(str1[3])
```

Operator

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

```
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.

```
print('Operator Aritmetika')
print(21+22) # 43
print(34-14) # 20
print(2*3) # 6
print(21/2) # 10.5
print(21.00/2.00) # 10.5
print(21%2) # 1
print(21.00//2.00) # 10.0
print(4**3) # 4 pangkat 3
print('Operator Relasional / Perbandingan')
print(3>22) # False
print(3<22) # True
print(4<=4) # True
print(4>=4) # True
print(5==5.0) # True
print(1!=1.0) # False
print('Operator Bitwise')
x = 25 # nilai awal
# 25 = 0001 1001
print(x >> 2) # 0000 0110 = 6
print(x << 2) # 0001 1000 = 24
a = 3 # 0000 0011
b = 6 # 0000 0110
# AND
print (a & b) # jika bit di dua operand sama, diaktifkan di hasil
               # 0000 0010 = 2
# 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
```

```
          # 0000 0111 = 7
x = 7
x^=2
print(x) # x = x lama XOR 2 = 7 xor 2
          # 7 = 0000 0111
          # 2 = 0000 0010
          # bit hidup jika di salah satu operand hidup,
          # tapi tidak di keduanya
          # 0000 0101 = 5
x = 7
x>>=2
print(x) # x = x lama >> 2 = 7 >> 2
          # 7 = 0000 0111
          # 0000 0001 = 1
x = 7
x<<=2
print(x) # x = x lama << 2 = 7 << 2
          # 7 = 0000 0111
          # 0001 1100 = 28
print('Operator Logika')
x = 3 > 1 and 2 < 19 # jika kedua sisi true -> true
print(x)
x = 3 > 4 or 2 < 10 # jika salah satu sisi benar -> true
print(x)
x = not(3 > 4) # not -> negasi
print(x)
print('Operator Keanggotaan / Membership')
x = (2,5,9,8,1,9,7,2)
print(9 in x)
print(10 in x)
print(10 not in x)
print('Operator Identitas / Identity')
x = 7
print(x is 7)
print(x is not 7)
```

Indentasi

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

```
a = (2,5,8,1,9,7,2)
```

```
for x in a:
    print(x)
    # harus ditulis dalam indentasi karena merupakan bagian kelompok
    # 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:

```
nilai = int(input("Masukkan nilai siswa = "))

if nilai > 60:
    print("Lulus")
else:
    print("Tidak lulus")

if nilai <= 10:
    print("Anda harus mengulang semester depan")

ipsemester = float(input("Masukkan nilai IP semester = "))

if ipsemester > 3:
    print("Anda bisa mengambil 24 SKS")
elif ipsemester >= 2.75:
    print("Anda bisa mengambil 21 SKS")
elif ipsemester >= 2:
    print("Anda bisa mengambil 18 SKS")
else:
    print("Anda hanya bisa mengambil 12 SKS")
```

while

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

```python
nilai = 1

# akan ditampilkan angka 1 - 9
# setelah itu berhenti
while nilai < 10:
    print(nilai)
    nilai += 1

while nilai <= 20:
    print(nilai)
    nilai += 1
else:
    print("nilai sudah mencapai 20 ...")

input("Tekan sembarang tombol untuk meneruskan ... ")

# akan ditampilkan angka 21
# dan seterusnya tidak akan berhenti
# kecuali ditekan Ctrl-C
while True:
    print(nilai)
    nilai += 1
```

for

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

```python
daftar = ["first", "2nd", 3]

for a in daftar:
    print(a)

for a in range(20):
    print(a) # 0 - 19

for a in range(1, 5):
    print(a) # 1 -4

for w in 'ABCDEFG':
    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:

```
def nama(arg1, arg2, ... , argN):
    ...
    Isi fungsi
    ...

# memanggil fungsi:

retvalnama = nama(arg1, arg2, ... , argN)
```

Contoh:

```
# function.py
def jumlah(arg1):
    jml = 0
    for a in arg1:
        jml += 1
    return jml

str_obj = "Wabi Teknologi Indonesia"
```

```
jml_str = jumlah(str_obj)

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.

```
# function_default.py
#
# menghitung jumlah karakter tertentu dalam string
# jika the_char tidak ada, maka default menghitung
# jumlah semua karakter

def jumlah(the_str, the_char=None):
    jml = 0
    if the_char:
        for a in the_str:
            if a == the_char:
                jml += 1
    else:
        for a in the_str:
            jml += 1
    return jml

str_obj = "Wabi Teknologi Indonesia"

jml_semua = jumlah(str_obj)
print(f'String {str_obj} mempunyai ' + str(jml_semua) + ' karakter')

jml_a = jumlah(str_obj, 'a')
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).

```python
# function_args_kwargs.py

def penambah(*args):
    total = 0
    for op in args:
        total += op
    return total

jml1 = penambah(1)
jml2 = penambah(23, 24, 21)
jml3 = penambah(1,2,3,4,5,6,7,8,9,10)

print(jml1)
print(jml2)
print(jml3)

def gabungkan(*args, pemisah='/'):
    return pemisah.join(args)

str_gabung = gabungkan('a','b','c')
str_gabung2 = gabungkan('a','b','c', pemisah='-')

print(str_gabung)
print(str_gabung2)

def get_kwargs(**kwargs):
    return kwargs

def get_key_value(**kwargs):
    for key, value in kwargs.items():
        print("Nilai {} = {}".format(key, value))

kw1 = get_kwargs(product_id='P001', product_name='Laptop')
print(kw1)
get_key_value(product_id='P001', product_name='Laptop')
```

Ekspresi / Operator / Fungsi Lambda

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.

```
# lambda1.py

# definisi:
# lambda arg1, arg2, ... , argN: ekspresi

multiple = lambda x, y: x * y

print(multiple(10,20))

def kali_berapa(n):
    return lambda a: a * n

kali_dua = kali_berapa(2)

print(kali_dua(90))
```

__main__

Saat script Python dipanggil / dieksekusi / dijalankan, variabel __name__ akan berisi nama modul. Jika file tersebut merupakan program utama (bukan modul), maka __name__ akan berisi __main__. Pada Python versi 2, hal ini bisa dideteksi dengan menggunakan baris:

```
if __name__ == "__main__":
```

Pada Python 3, langung memanggil *main()*.

```
def jumlah(arg1):
    jml = 0
    for a in arg1:
        jml += 1
    return jml

def main():
```

```
    str_obj = "Wabi Teknologi Indonesia"
    jml_str = jumlah(str_obj)

    print(f'String {str_obj} mempunyai ' + str(jml_str) + ' karakter')

#if __name__== "__main__":
#    main()

# Menggunakan Python 3 lebih singkat:
# langsung panggil main()

main()
```

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.

```
# list.py

daftar_makanan = ['soto', 'bakso', 'pecel', 'nila bakar']

print(daftar_makanan)
# hasil: ['soto', 'bakso', 'pecel', 'nila bakar']

print()
for makanan in daftar_makanan:
    print(makanan)
    # hasil:
    #   soto
    #   bakso
    #   pecel
    #   nila bakar

print()
print(daftar_makanan[0])
# soto

print()
print(daftar_makanan[-1])
# nila bakar

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.

```
# set.py

proglang = {'Rust', 'Python', 'Go', 'Rust'}
print(proglang)
tambahan = ('Ruby', 'Lua')
proglang.add(tambahan)
print(proglang)
print('Rust' in proglang)

huruf = set('Wabi Teknologi')
print(huruf)

huruf2 = set()
print(huruf2)
huruf2.add('Wabi Teknologi')
print(huruf2)

kata1 = set('indonesia')
kata2 = set('merdeka')
print(kata1)
print(kata2)

# ada di kata1, tidak ada di kata2
print(kata1 - kata2)

# ada di kata1 atau di kata2 atau di keduanya
print(kata1 | kata2)

# ada di kata1 dan kata2
print(kata1 & kata2)

# ada di kata1 atau di kata2 tapi tidak di keduanya
print(kata1 ^ kata2)
```

Lihat juga help(set).

Dictionary

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

```
# dict.py

rumah = {'H-304': 'Bambang Purnomosidi', 'H-303': 'Anton'}
print(rumah)
print(rumah.items())
print(rumah['H-304'])
for k, v in rumah.items():
    print(f'Rumah nomor {k} adalah tempat tinggal keluarga {v}')

print('H-304' in rumah)
print('H-305' in rumah)
print('H-304' not in rumah)
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.

```
# modul-01.py

def penambah(*args):
    total = 0
    for op in args:
        total += op
    return total

# jika di-import, maka __name__ berisi nama modul yaitu
# namafile.py
# jika dijalankan dari shell / command line, maka
# __name__ akan berisi '__main__'
# jadi, di bawah ini tidak akan dijalankan jika di-import
if __name__ == '__main__':
    print(penambah(32,43,12))
```

Jika dipanggil dari command line / shell:

```
» python modul01.py
87
```

Jika di-import:

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

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:

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

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.

```
» pip list
Package                  Version
------------------------ ----------
alabaster                0.7.11
anytree                  2.4.3
appdirs                  1.4.3
Babel                    2.6.0
backcall                 0.1.0
Brlapi                   0.6.7
btrfsutil                1.0.0
```

```
CacheControl              0.12.5
chardet                   3.0.4
colorama                  0.4.1
decorator                 4.3.0
distlib                   0.2.8
distro                    1.3.0
docutils                  0.14
greenlet                  0.4.15
html5lib                  1.0.1
...
...
```

Untuk menginstall paket, gunakan:

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

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:

```
conda list => daftar paket yang sudah terinstall
conda install x => install paket x
conda remove x => uninstall paket x
conda update x => update paket x
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

```
# input_keyboard.py

nama = input('Masukkan nama = ')
usia = int(input('Umur = '))

print(f'Nama = {nama}, usia = {usia}')
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().

```
# open_file_with.py

# default: tanpa argumen mode, dibuka utk dibaca (read)
with open('angka.txt') as f:
    read_data = f.read()
    print(read_data)
    # error: io.UnsupportedOperation: not writable
```

```
    #f.write('tambahan 1')

# dengan 'with',  tidak perlu di close
print(f.closed)

# r+ => read write
with open('angka.txt','r+') as f:
    read_data = f.read()
    print(read_data)
    # bisa ditulisi karena r+
    f.write('tambahan')

# r+ => read write
# dibuka dengan w+ membuat isi hilang
with open('angka.txt','w') as f:
    f.write('tambahan')

with open('angka.txt') as f:
    read_data = f.read()
    print(f'sekarang hanya berisi 1 baris: {read_data}')

# sekarang diisi dengan loop angka
with open('angka.txt', 'w') as f:
    for a in range(1,11):
        f.write(str(a) + '\n')

# tampilkan hasil pengisian
with open('angka.txt') as f:
    for line in f:
        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):

```
>>> f = open('abc.txt')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
FileNotFoundError: [Errno 2] No such file or directory: 'abc.txt'
>>> f = open('abca.txt','f')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid mode: 'f'
>>> print 'abcdefg'
  File "<stdin>", line 1
    print 'abcdefg'
                 ^
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.

```
# except.py

```

```python
import sys

# tanpa exception handling
# b = float(input('masukkan angka float: '))
# amasukkan angka float: f
# Traceback (most recent call last):
#  File "except.py", line 5, in <module>
#    b = float(input('masukkan angka float: '))
# ValueError: could not convert string to float: 'f'

# Setelah dihandle:
try:
    a = float(input("masukkan angka float: "))
except ValueError:
    print('harus memasukkan nilai float')

# Jika tidak tau kemungkinan error:
try:
    a = float(input("masukkan angka: "))
    b = float(input("masukkan angka pembagi: "))
    z = a/b
except:
    print("Error:", sys.exc_info()[0])
else:
    print('Hasil bagi: ', z)
finally:
    # bagian ini biasanya untuk clean-up, di dunia nyata
    # biasanya berisi bagian utk close connection, menutup
    # file dan lain-lain
    print('Selesai')
# hasil:
# masukkan angka: 40
# masukkan angka pembagi: 0
# 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  bpdp = Dosen()
9  bpdp.nama = 'Bambang Purnomosidi'
10
11 print(bpdp)
12 print(bpdp.nama)
13
14 class DosenSTMIKAkakom(Dosen):
15
16     institusi = 'STMIK AKAKOM'
17
18     # konstruktor
```

```
    def __init__(self, npp, nama):
        self.npp = npp
        self.nama = nama

    def mengajar(self, *args):
        self.mk_diampu = args

bambangpdp = DosenSTMIKAkakom('123', 'bpdp')
print(bambangpdp)
bambangpdp.mengajar('Teknologi Cloud Computing', 'Big Data Analytics')
print(bambangpdp.mk_diampu)
print(bambangpdp.institusi)

class DosenSTMIKAkakomTI(DosenSTMIKAkakom):

    prodi = 'Teknik Informatika'

nia = DosenSTMIKAkakomTI('213','Nia R')
print(nia.institusi)
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).

```
# non_pure_function.py

a = 200

def change_state():

    global a

    a += 100

    return a

print(change_state())
print(change_state())
print(change_state())
print(change_state())
print(change_state())
```

Untuk pure function, silahkan lihat contoh berikut:

```
# 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.

```
# generator.py

def generate_val(N):
    for i in range(N):
        yield i

hasil = generate_val(10)
print(hasil)

for a in hasil:
    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.

```
# map.py

def make_ucase(the_str):
    return the_str.upper()

# a => iterable
a = ['a', 'b', 'c']

# kerjakan fungsi make_ucase utk setiap item a
b = map(make_ucase,a)

for c in b:
    print(c)
```

Reduce

Reduce digunakan untuk mengubah obyek iterable menjadi satu nilai saja.

```
# reduce.py

from functools import reduce

# tanpa lambda expression dan reduce
hasil = 1
x = [1, 2, 3, 4, 5]
for num in x:
    hasil = hasil * num

print(hasil)

# dengan lambda expression dan reduce
hasil2 = reduce((lambda x, y: x * y),[1, 2, 3, 4, 5])

print(hasil2)

# hasil:
# 120
# 120
```

Filter

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

```
# filter.py

nilai = range(-10, 10)

for a in nilai:
    print(a)
    # hasilL -10 sampai 10

# Kita akan memfilter list sehingga hanya yang berisi nilai positif
# yang akan masuk ke list baru

```

```python
l_baru = list(filter(lambda angka: angka > 0, nilai))
print(l_baru)
# 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.

```python
# hof.py (higher order function)

# HOF - fungsi sebagai argumen fungsi
def penjumlahan(angka):
    return sum(angka)

def aksi(func, angka2):
    return func(angka2)

print(aksi(penjumlahan, [1, 2, 3, 4, 5]))

# HOF - fungsi sebagai return value
def remaja():
    return "remaja"
def dewasa():
    return "dewasa"

def person():
    umur = int(input("Umur anda: "))

    if umur <= 21:
        return remaja()
    else:
        return dewasa()

print(person())
```

Closure

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

```python
# closure.py

from functools import partial

# bilangan pangkat eksponen
def pangkat(bilangan, eksponen):
  return bilangan ** eksponen

kuadrat = partial(pangkat, eksponen=2)
print(kuadrat(2))
# hasil = 2

# parsial:
# pangkat dipanggil dengan arg eksponen ditetapkan di awal
pangkat_empat = partial(pangkat, eksponen=4)
print(pangkat_empat(2))
# hasil = 16
```

Asynchronous I/O / Concurrent Programming di Python

Concurrent programming adalah bentuk komputasi yang memungkinan 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.

```
# asynchronous.py

# diambil dari manual Python
# https://docs.python.org/3/library/asyncio-task.html

import asyncio

async def factorial(name, number):
    f = 1
    for i in range(2, number + 1):
        print(f"Task {name}: Compute factorial({i})...")
        await asyncio.sleep(1)
        f *= i
    print(f"Task {name}: factorial({number}) = {f}")

async def main():
    # Schedule three calls *concurrently*:
    await asyncio.gather(
        factorial("A", 8),
        factorial("B", 3),
```

```
        factorial("C", 4),
    )

asyncio.run(main())
```