ZimeraSchool

ZimeraCorp

A¬¬o¬¬¬¬¬, S¬¬¬¬¬¬¬ D¬¬¬, ¬¬¬ P¬¬¬¬¬¬¬¬ P¬¬¬o¬¬¬¬¬¬ M¬¬¬¬¬¬¬¬¬¬ P¬¬¬o¬

Dr. Bambang Purnomosidi D. P. <bambangpdp@gmail.com>

Version 0.0.1-rev-2022-12-05 07:10:33 +0700

Daftar Isi

Kata Pengantar

Selamat datang di buku Algoritam, Struktur Data, dan Paradigma Pemrograman Menggunakan Python. Buku ini merupakan buku yang digunakan di Zimera School sebagai buku pegangan untuk para pembelajar dalam mempelajari Python. Silakan menggunakan buku ini dan diijinkan untuk menyebarkan format PDF dari buku ini ke siapa saja tanpa mengedit isinya. Terima kasih.

Bagian I: Ekosistem Bahasa Pemrograman Python

Bagian I membahas tentang pengenalan bahasa pemrograman Python serta ekosistem yang melingkupi bahasa pemrograman Python. Pembahasan tentang hal ini diperlukan untuk mendapatkan gambaran garis besar dari bahasa pemrograman Python serta berbagai masalah pemrograman yang bisa diselesaikan menggunakan Python dan penyiapan berbagai perangkat untuk pengembangan software menggunakan bahasa pemrograman Python.

Chapter 1. Mengenal Bahasa Pemrograman dan Python

1.1. Sistem Komputer dan Rangkaian Instruksi ke Komputer

Suatu sistem komputer merupakan serangkaian peranti keras / *hardware / devices* yang terintegrasi dan dikendalikan oleh peranti lunak / *software* untuk melaksanakan proses komputasi tertentu. Peranti keras terdiri atas peranti untuk masukan, pemroses, serta keluaran. Secara umum, sistem komputer akan mengambil masukan, memproses masukan tersebut, kemudian menghasilkan sesuatu dari pemrosesan tersebut. Perhatikan misal saat orang akan menghasilkan dokumen tercetak menggunakan sistem komputer:

- Masukan: dokumen yang akan dibuat dalam format tercetak, dimasukkan / ditulis menggunakan *keyboard* pada sistem komputer.
- Pemroses: *software* yang dgunakan untuk memproses dokumen, misalnya LibreOffice Writer.
- Keluaran: hasil ketikan tersebut kemudian dicetak ke peranti keluaran - *printer* - menggunakan salah satu fasilitas dari LibreOffice.

Untuk membuat supaya berbagai *hardware* tersebut bisa mengerjakan sesuai dengan yang diinstruksikan, maka sistem komputer harus diprogram sesuai dengan tugas / *task* yang dikehendaki. Hasil dari proses memprogram tersebut adalah *software*. *Software* sering juga disebut dengan istilah *program*.

1.2. *Software*

Suatu sistem komputer dikendalikan oleh *software*. Tanpa *software*, sistem komputer tidak akan lengkap dan tidak bisa melakukan aktivitas apapun. Ada beberapa kategori *software*:

1. Sistem Operasi / Sistem Pengoperasi / *Operating System*. Merupakan *software* paling mendasar yang mutlak harus dimiliki oleh sistem komputer. Tanpa sistem operasi, *software* lainnya tidak akan bisa berjalan. Sistem operasi berfungsi untuk mengelola berbagai sumber daya komputasi dan menyediakan abstraksi serta perangkat dasar komputasi ke para pemakai. Sebagai contoh, supaya bisa menggunakan peranti printer, maka *software* harus mengakses abstraksi yang dibuat oleh sistem operasi dalam bentuk API (*Applicaton Programming Interface*) untuk printer. *Software* yang akan menggunakan printer tidak perlu mengakses *hardware* secara langsung. Selain itu, sistem operasi juga menyediakan *software* dasar untuk keperluan komputasi, misal Windows mempunyai Notepad untuk membuat suatu file teks, Linux mempunyai *shell* untuk memberikan perintah ke sistem operasi, dan lain-lain.

Contoh dari sistem operasi adalah Windows, Linux, FreeBSD, NetBSD, dan lain-lain.

2. Peranti Pengembangan / *Development Tools*. *Software* ini merupakan *software* yang digunakan untuk membangun *software* dan menjadi perhatian utama dari para pemrograman untuk menghasilkan *software*. Beberapa kategori peranti pengembangan ini antara lain adalah kompilator / *interpreter* untuk bahasa pemrograman tersebut, *profiler*, *debugger*, IDE (*Integrated Development Environment*), dan lain-lain. Contoh peranti pengembangan: GNU C, LLVIM Clang, CPython, Rust Compiler, Go Compiler, GNU Debugger, Vim, Neovim, Emacs, Visual Studio, Visual Studio Code, dan lain-lain.
3. *Software* Aplikasi. *Software* ini dibuat untuk tugas khusus yang diperlukan oleh *end user*, misalnya LibreOffice (untuk pekerjaan mengolah berbagai dokumen perkantoran), Adobe Photoshop (untuk membuat / mengedit / mengelola gambar), GIMP (untuk membuat / mengedit / mengelola gambar), OBS (untuk membuat video tutorial), dan lain-lain.
4. *Software Utilitas*. *Software* ini dibuat untuk keperluan meningkatkan kinerja sistem, baik di level sistem operasi maupun aplikasi. Contoh: Defragementer di Windows digunakan untuk melakukan pengurutan konten file fi media penyimpan sehingga isinya lebih cepat diakses, Partition Magic untuk mengelola partisi harddisk, Anti Virus, *shell utilities* di Linux untuk berbagai keperluan administrasi sistem, dan lain-lain.

1.3. Peranti Pengembangan

Peranti pengembangan digunakan untuk membuat *software* atau program, dengan menggunakan peranti ini, pemrogram memberikan instruksi yang harus dikerjakan oleh komputer. Untuk membuat *software*, seorang pemrogram harus menggunakan bahasa pemrograman tertentu, membuat kode sumber (instruksi dalam bahasa pemrograman tertentu), kemudian menjalankan kode sumber tersebut menggunakan kompilator / *interpreter* bahasa pemrograman tersebut. Jika terdapat kesalahan, maka akan terjadi error dan eksekusi kode sumber tersebut gagal. Pemrogram kemudian mencari penyebab dari error tersebut dan kemudian akan membetulkan dan kemudian mengeksekusi / menjalankan lagi sampai mendapatkan hasil yang diinginkan. Demikian seterusnya kurang lebih aktivitas yang dilakukan oleh pemrogram. Secara umu, aktivitas tersebut kan digambarkan sebagai berikut:

1. Temukan masalah
2. Buat dan perbaiki algoritma
3. Implementasikan algoritma dalam bentuk kode sumber
4. Jalankan kode sumber menggunakan kompilator / *interpreter*.

5. Jika sudah sesuai, berhenti. Jika belum, kembali ke langkah 2.

Untuk melakukan berbagai aktivitas tersebut, pemrogram memerlukan berbagai *software*. Secara minimum, pemrogram akan memerlukan:

1. Kompilator / *interpreter* bahasa pemrograman
2. Editor untuk menuliska kode sumber

Semakin kompleks masalah yang ingin dicari solusinya, semakin kompleks *software* yang dibutuhkan. Beberapa tambahan tersebut diantaranya adalah sebagai berikut:

1. *Build tools*
2. *Profiler*
3. *Debugger*
4. *Libraries*
5. *Security Tools*
6. *CI (Continuous Integration) - CD (Continuous Delivery)*
7. *Interface Designer*
8. IDE (*Integrated Development Environment*)
9. *Database Tools*
10. *Code coverage Tools*
11. *Documentation Tools*
12. *Revision Control Sofware*

1.4. Bahasa Pemrograman: Spesifikasi dan Implementasi

Suatu sistem bisa diberikan instruksi melalui bahasa pemrograman dan implementasi bahasa pemrograman. Untuk keperluan tersebut, bisa dibedakan antara bahasa pemrograman dengan implementasi bahasa pemrograman tersebut. Setiap bahasa pemrograman mempunyai spesifikasi sintaksis dan berdasarkan sintaksis tersebut dibuat *compiler / kompilator* atau *interpreter* yang akan menterjemahkan sintaksis bahasa pemrograman tersebut ke dalam bahasa yang dipahami oleh mesin sehingga bisa dijalankan / dieksekusi oleh sitem komputer tersebut. Contoh spesifikasi dan implementasi:

1. Java: spesifikasi ada di https://docs.oracle.com/javase/specs/, implementasi: Oracle JDK, Eclipse Temurin.
2. Python: spesifikasi ada di https://docs.python.org/dev/reference/index.html, implentasi: CPython, IronPython, Jython.
3. JavaScript: spesifikasi dibuat oleh tim di ECMAScript (https://www.ecma-international.org/technical-committees/tc39/), implementasi: Node.js, Bun, Deno

Meskipun kebanyakan membedakan bahasa pemrograman dengan implementasi, ada juga beberapa yang menjadikan satu sehingga penyebutan bahasa pemrograman sekaligus sebagai penyebutan *kompilator* / *interpreter* (kondisi ini biasanya untuk bahasa pemrograman yang masih dalam taraf perkembangan atau tidak dimaksudkan untuk diimplementasikan oleh berbagai vendor dengan menggunakan spesifikasi terbuka), sebagai contoh Zig (https://ziglang.org/),

1.5. Apakah Python Itu?

Python adalah spesifikasi bahasa pemrograman serta peranti penerjemah (**interpreter**) untuk menjalankan / mengeksekusi *source code* / kode sumber 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.

1.6. 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.

1.7. 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) - https://www.jython.org/
- **Pyston** (Python yang dimplementasikan dengan teknik JIT) - https://www.pyston.org/
- **RustPython** (Python yang diimplementasikan menggunakan bahasa pemrograman Rust) - https://rustpython.github.io/
- **IronPython** (Python di .NET) - https://ironpython.net/
- **Stackless** (Python dengan microthreads - threads yang tidak dikelola oleh OS, tetapi dikelola oleh Stackless) - http://www.stackless.com/
- **MicroPython** (Python untuk microcontroller) - https://micropython.org/
- **CircuitPython** (Turunan dari MicroPython untuk siswa dan pemula) - https://circuitpython.org/
- **PyPy** (Python JIT Compiler) - https://www.pypy.org/

- **Anaconda** (Python standar yang sudah menyertakan conda) - https://anaconda.org/
- **Intel Distribution for Python** (Distribusi Python yang dibuat oleh Intel) - https://software.intel.com/en-us/python-distribution).

Materi di buku 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 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/.

Chapter 2. Persiapan Peranti Pengembangan

2.1. 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.

2.2. 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:

```
$ wget https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-
x86_64.sh
--2022-10-03 11:41:09--  https://repo.anaconda.com/miniconda/Miniconda3-
latest-Linux-x86_64.sh
Resolving repo.anaconda.com (repo.anaconda.com)... 2606:4700::6810:8303,
2606:4700::6810:8203, 104.16.130.3, ...
Connecting to repo.anaconda.com (repo.anaconda.com
)|2606:4700::6810:8303|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 76607678 (73M) [application/x-sh]
Saving to: Miniconda3-latest-Linux-x86_64.sh

Miniconda3-latest-Linux-x86_64.sh
100%[=====================================================================
=>]  73.06M  2.24MB/s    in 32s

2022-10-03 11:41:41 (2.29 MB/s) - Miniconda3-latest-Linux-x86_64.sh
```

```
saved [76607678/76607678]
$ ls -la
total 74828
drwxr-xr-x   2 bpdp bpdp     4096 Oct  3 11:41 ./
drwxr-xr-x 167 bpdp bpdp     4096 Sep 15 21:42 ../
-rw-r--r--   1 bpdp bpdp 76607678 May 17 03:01 Miniconda3-latest-Linux-
x86_64.sh
$ chmod +x Miniconda3-latest-Linux-x86_64.sh
$ ls -la
total 74828
drwxr-xr-x   2 bpdp bpdp     4096 Oct  3 11:41 ./
drwxr-xr-x 167 bpdp bpdp     4096 Sep 15 21:42 ../
-rwxr-xr-x   1 bpdp bpdp 76607678 May 17 03:01 Miniconda3-latest-Linux-
x86_64.sh*
$ ./Miniconda3-latest-Linux-x86_64.sh

Welcome to Miniconda3 py39_4.12.0

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

Copyright 2015-2022, Anaconda, Inc.

All rights reserved under the 3-clause BSD License:

This End User License Agreement (the "Agreement") is a legal agreement
between you and Anaconda, Inc. ("Anaconda") and governs your use of
Miniconda.

....
....
....

Do you accept the license terms? [yes|no]
[no] >>> yes

Miniconda3 will now be installed into this location:
/home/bpdp/miniconda3

  - Press ENTER to confirm the location
  - Press CTRL-C to abort the installation
  - Or specify a different location below

[/home/bpdp/miniconda3] >>>
PREFIX=/home/bpdp/miniconda3
Unpacking payload ...
Collecting package metadata (current_repodata.json): done
```

```
Solving environment: done

## Package Plan ##

  environment location: /home/bpdp/miniconda3

  added / updated specs:
    - _libgcc_mutex==0.1=main
    - _openmp_mutex==4.5=1_gnu
    - brotlipy==0.7.0=py39h27cfd23_1003
    - ca-certificates==2022.3.29=h06a4308_1
    - certifi==2021.10.8=py39h06a4308_2
    - cffi==1.15.0=py39hd667e15_1
    - charset-normalizer==2.0.4=pyhd3eb1b0_0
    - colorama==0.4.4=pyhd3eb1b0_0
    - conda-content-trust==0.1.1=pyhd3eb1b0_0
    - conda-package-handling==1.8.1=py39h7f8727e_0
    - conda==4.12.0=py39h06a4308_0
    - cryptography==36.0.0=py39h9ce1e76_0
    - idna==3.3=pyhd3eb1b0_0
    - ld_impl_linux-64==2.35.1=h7274673_9
    - libffi==3.3=he6710b0_2
    - libgcc-ng==9.3.0=h5101ec6_17
    - libgomp==9.3.0=h5101ec6_17
    - libstdcxx-ng==9.3.0=hd4cf53a_17
    - ncurses==6.3=h7f8727e_2
    - openssl==1.1.1n=h7f8727e_0
    - pip==21.2.4=py39h06a4308_0
    - pycosat==0.6.3=py39h27cfd23_0
    - pycparser==2.21=pyhd3eb1b0_0
    - pyopenssl==22.0.0=pyhd3eb1b0_0
    - pysocks==1.7.1=py39h06a4308_0
    - python==3.9.12=h12debd9_0
    - readline==8.1.2=h7f8727e_1
    - requests==2.27.1=pyhd3eb1b0_0
    - ruamel_yaml==0.15.100=py39h27cfd23_0
    - setuptools==61.2.0=py39h06a4308_0
    - six==1.16.0=pyhd3eb1b0_1
    - sqlite==3.38.2=hc218d9a_0
    - tk==8.6.11=h1ccaba5_0
    - tqdm==4.63.0=pyhd3eb1b0_0
    - tzdata==2022a=hda174b7_0
    - urllib3==1.26.8=pyhd3eb1b0_0
    - wheel==0.37.1=pyhd3eb1b0_0
    - xz==5.2.5=h7b6447c_0
    - yaml==0.2.5=h7b6447c_0
    - zlib==1.2.12=h7f8727e_1


The following NEW packages will be INSTALLED:

  _libgcc_mutex      pkgs/main/linux-64::_libgcc_mutex-0.1-main
```

```
  _openmp_mutex      pkgs/main/linux-64::_openmp_mutex-4.5-1_gnu
  brotlipy           pkgs/main/linux-64::brotlipy-0.7.0-py39h27cfd23_1003
  ca-certificates    pkgs/main/linux-64::ca-certificates-2022.3.29-
h06a4308_1
  certifi            pkgs/main/linux-64::certifi-2021.10.8-py39h06a4308_2
  cffi               pkgs/main/linux-64::cffi-1.15.0-py39hd667e15_1
  charset-normalizer pkgs/main/noarch::charset-normalizer-2.0.4-
pyhd3eb1b0_0
  colorama           pkgs/main/noarch::colorama-0.4.4-pyhd3eb1b0_0
  conda              pkgs/main/linux-64::conda-4.12.0-py39h06a4308_0
  conda-content-tru~ pkgs/main/noarch::conda-content-trust-0.1.1-
pyhd3eb1b0_0
  conda-package-han~ pkgs/main/linux-64::conda-package-handling-1.8.1-
py39h7f8727e_0
  cryptography       pkgs/main/linux-64::cryptography-36.0.0-
py39h9ce1e76_0
  idna               pkgs/main/noarch::idna-3.3-pyhd3eb1b0_0
  ld_impl_linux-64   pkgs/main/linux-64::ld_impl_linux-64-2.35.1-
h7274673_9
  libffi             pkgs/main/linux-64::libffi-3.3-he6710b0_2
  libgcc-ng          pkgs/main/linux-64::libgcc-ng-9.3.0-h5101ec6_17
  libgomp            pkgs/main/linux-64::libgomp-9.3.0-h5101ec6_17
  libstdcxx-ng       pkgs/main/linux-64::libstdcxx-ng-9.3.0-hd4cf53a_17
  ncurses            pkgs/main/linux-64::ncurses-6.3-h7f8727e_2
  openssl            pkgs/main/linux-64::openssl-1.1.1n-h7f8727e_0
  pip                pkgs/main/linux-64::pip-21.2.4-py39h06a4308_0
  pycosat            pkgs/main/linux-64::pycosat-0.6.3-py39h27cfd23_0
  pycparser          pkgs/main/noarch::pycparser-2.21-pyhd3eb1b0_0
  pyopenssl          pkgs/main/noarch::pyopenssl-22.0.0-pyhd3eb1b0_0
  pysocks            pkgs/main/linux-64::pysocks-1.7.1-py39h06a4308_0
  python             pkgs/main/linux-64::python-3.9.12-h12debd9_0
  readline           pkgs/main/linux-64::readline-8.1.2-h7f8727e_1
  requests           pkgs/main/noarch::requests-2.27.1-pyhd3eb1b0_0
  ruamel_yaml        pkgs/main/linux-64::ruamel_yaml-0.15.100-
py39h27cfd23_0
  setuptools         pkgs/main/linux-64::setuptools-61.2.0-py39h06a4308_0
  six                pkgs/main/noarch::six-1.16.0-pyhd3eb1b0_1
  sqlite             pkgs/main/linux-64::sqlite-3.38.2-hc218d9a_0
  tk                 pkgs/main/linux-64::tk-8.6.11-h1ccaba5_0
  tqdm               pkgs/main/noarch::tqdm-4.63.0-pyhd3eb1b0_0
  tzdata             pkgs/main/noarch::tzdata-2022a-hda174b7_0
  urllib3            pkgs/main/noarch::urllib3-1.26.8-pyhd3eb1b0_0
  wheel              pkgs/main/noarch::wheel-0.37.1-pyhd3eb1b0_0
  xz                 pkgs/main/linux-64::xz-5.2.5-h7b6447c_0
  yaml               pkgs/main/linux-64::yaml-0.2.5-h7b6447c_0
  zlib               pkgs/main/linux-64::zlib-1.2.12-h7f8727e_1


Preparing transaction: done
Executing transaction: done
installation finished.
Do you wish the installer to initialize Miniconda3
```

```
by running conda init? [yes|no]
[no] >>>

You have chosen to not have conda modify your shell scripts at all.
To activate conda's base environment in your current shell session:

eval "$(/home/bpdp/miniconda3/bin/conda shell.YOUR_SHELL_NAME hook)"

To install conda's shell functions for easier access, first activate,
then:

conda init

If you'd prefer that conda's base environment not be activated on startup,
   set the auto_activate_base parameter to false:

conda config --set auto_activate_base false

Thank you for installing Miniconda3!
$
```

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

```
set -x PATH /home/bpdp/miniconda3/bin $PATH
source /home/bpdp/miniconda3/etc/fish/conf.d/conda.fish
```

Jika menggunakan shell **Bash**:

```
export PATH /home/bpdp/miniconda3/bin:PATH
source /home/bpdp/miniconda3/etc/profile.d/conda.sh
```

Setelah itu, *interpreter* Python dan **conda** bisa dijalankan:

```
$ python
Python 3.9.12 (main, Apr  5 2022, 06:56:58)
[GCC 7.5.0] :: Anaconda, Inc. on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Untuk menjalankan conda:

```
$ conda
```

```
usage: conda [-h] [-V] command ...

conda is a tool for managing and deploying applications, environments and
packages.

Options:

positional arguments:
  command
    clean        Remove unused packages and caches.
    compare      Compare packages between conda environments.
    config       Modify configuration values in .condarc. This is modeled
after the git config command. Writes to the user .condarc file
                 (/home/bpdp/.condarc) by default.
    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.
    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:
  content-trust
  env
```

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.

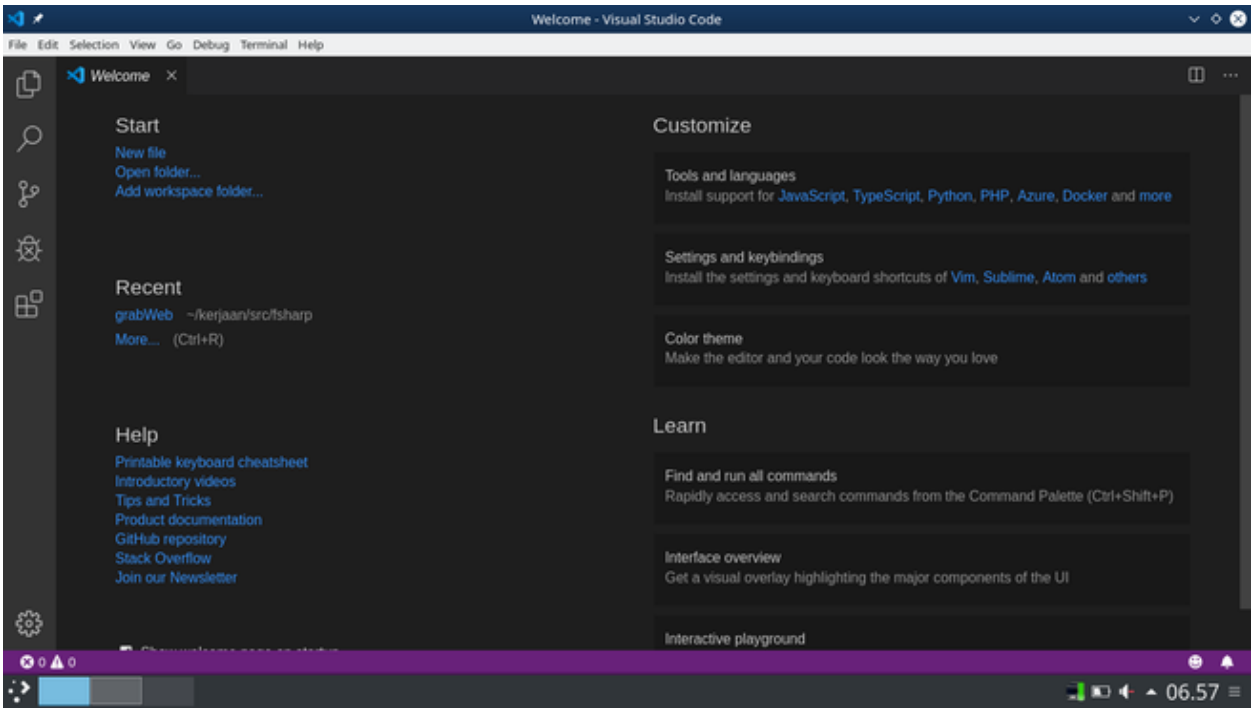
2.3. Peranti Pendukung: IDE atau Editor Teks

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.

2.4. 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 seperti yang terdapat pada Tampilan awal dari Visual Studio Code.



Gambar 1. Tampilan awal dari Visual Studio Code

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

2.5. Instalasi Extension

Extension Python untuk VSCode tersedia di **marketplace** pada URL https://marketplace.visualstudio.com/items?itemName=ms-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:

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

Chapter 3. Algoritma, Pemrograman, Paradigma Pemrograman, dan Python

3.1. Pengertian Algoritma

Istilah **Algoritma** berasal dari nama matematikawan abad 9, yaitu * Mu¬ammad ibn M¬s¬ al-Khw¬rizm¬* (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.

3.2. 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.

3.3. 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
```

3.4. 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.

3.5. Python dan Implementasi Algoritma

Python merupakan bahasa pemrograman yang mempunyai sintaksis atau tata bahasa yang menyerupai **pseudocode** sehingga relatif mudah 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. Gunakan sintaks Python untuk mengimplementasikan langkah per langkah dari algoritma.

Contoh dari proses tersebut adalah sebagai berikut:

1. Definisi masalah

```
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
```

```
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, implementasikan algoritma tersebut:

src/01-03/angka-max/01-max.py

```
angka1 = input("Masukkan angka pertama: ")              ①
angka2 = input("Masukkan angka kedua: ")

if angka1 > angka2:                                     ②
  print("Nilai terbesar: ", angka1)                     ③
elif angka1 < angka2:
  print("Nilai terbesar: ", angka2)
elif angka1 == angka2:
  print("Kedua angka sama, tidak ada yang terbesar")
else:
  print("Ada masalah dengan angka masukan anda")
```

① Perintah **input** digunakan untuk meminta masukan melalui keyboard

② **if** digunakan untuk memeriksa ekpresi dan melakukan sesuatu berdasarkan hasil pemeriksaan. **(angka1 > angka2)** adalah bentuk ekspresi.

③ **print** pada bagian ini adalah perintah yang akan dikerjakan jika hasil pemeriksaan ekspresi menghasilkan nilai benar (*True*).

Untuk menjalankan:

```
$ python max.py
Masukkan angka pertama: 43
Masukkan angka kedua: 12
Nilai terbesar:  43
$ python max.py
Masukkan angka pertama: 11
Masukkan angka kedua: 11
Kedua angka sama, tidak ada yang terbesar
$ python max.py
Masukkan angka pertama: abc
Masukkan angka kedua: def
Nilai terbesar:  def
$
```

Setelah algoritma diimplementasikan dan kemudian kode sumber bisa dijalankan, maka seringkali muncul masalah. Sebagai contoh, kode sumber di atas sudah bisa dijalankan dengan baik, tetapi jika diisikan bukan angka, maka kode sumber tetap berjalan dan memberikan hasil. Hal ini tentu saja tidak diinginkan. Apa yang harus dilakukan? secara sederhana, pemrogram kemudian berpikir bahwa **berarti harus ada mekanisme untuk memeriksa apakah angka yang dimasukkan tersebut angka atau bukan**. Setelah mempunyai pikiran seperti ini, pemrogram kemudian mencari cara (di manual, di Internet, bertanya ke rekan-rekan, dan lain-lain) untuk memeriksa apakah input yang dimasukkan tersebut angka atau bukan. Dari proses pencarian, diperoleh suatu fungsi untuk memeriksa apakah angka atau bukan, yaitu **isnumeric**. Kode sumber kemudian kita ubah untuk keperluan tersebut.

src/01-03/angka-max/02-max-check-numeric.py

```
angka1 = input("Masukkan angka pertama: ")
angka2 = input("Masukkan angka kedua: ")

if angka1.isnumeric() and angka2.isnumeric():                    ①
    if angka1 > angka2:
        print("Nilai terbesar: ", angka1)
    elif angka1 < angka2:
        print("Nilai terbesar: ", angka2)
    elif angka1 == angka2:
        print("Kedua angka sama, tidak ada yang terbesar")
    else:
        print("Ada masalah dengan angka masukan anda")
else:
    print("Anda tidak memasukkan angka")
```

① Memeriksa apakah isi angka1 dan angka2 numerik atau bukan dengan **isnumeric()**

Untuk menjalankan:

```
$ python 02-max-check-numeric.py
Masukkan angka pertama: abc
Masukkan angka kedua: def
Anda tidak memasukkan angka
$
```

Bagian II: Sintaksis Bahasa Pemrograman Python

Bagian II membahas tentang sisi teknis sintaksis bahasa pemrograman Python serta implementasi berbagai paradigma pemrograman menggunakan Python.

Chapter 4. Mulai Menggunakan Python

Python bisa digunakan dengan 2 cara:

1. REPL (**Read-Eval-Print-Loop**)
2. **Coding** - membuat kode sumber dalam bahasa pemrograman Python dan kemudian di-eksekusi / dijalankan menggunakan **interpreter** Python.

4.1. REPL

REPL digunakan untuk keperluan mencoba potongan kode sumber. Untuk keluar dari REPL maupun dari **help**>, gunakan **Ctrl-D**

```
$ python
Python 3.10.4 (main, Mar 31 2022, 08:41:55) [GCC 7.5.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> help()

Welcome to Python 3.10'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/3.10/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> if

The "if" statement
******************

The "if" statement is used for conditional execution:

   if_stmt ::= "if" assignment_expression ":" suite
               ("elif" assignment_expression ":" suite)*
               ["else" ":" suite]

It selects exactly one of the suites by evaluating the expressions one
by one until one is found to be true (see section Boolean operations
for the definition of true and false); then that suite is executed
```

```
(and no other part of the "if" statement is executed or evaluated).
If all expressions are false, the suite of the "else" clause, if
present, is executed.

Related help topics: TRUTHVALUE
(END)

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.

4.2. 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:

1. Cara 1

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

2. Cara 2

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

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

3. Cara 3

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

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

Chapter 5. Dasar-dasar Python

5.1. Identifier / Nama

Saat memprogram menggunakan Python, seorang pemrogram 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:

```
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
```

5.2. 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
"""
```

5.3. 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 = 'Zimera Corp'
>>> 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.

5.3.1. 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 *integer* (bulat):

src/02-02/numeric_int.py

```
# Integer

print(9834598304985034850394850348503458345 + 1)
# Hasil:
# 9834598304985034850394850348503458346
```

Bilangan pecahan (*floating point*):

src/02-02/numeric_float.py

```
print(10/3)
print(10/3.0)
print(type(10/3))
print(type(10/3.0))
# Hasil:
# 3
# 3.33333333333
# <type 'int'>
# <type 'float'>

# Notasi saintifik (e atau E)
print(+1e6)
print(1e-4)
print(.4e6)
print(5.6e-3)
# Hasil:
# 1000000.0
# 0.0001
# 400000.0
# 0.0056
```

Bilangan kompleks:

src/02-02/numeric_complex.py

```
x = 6
y = 4

z = complex(x,y);

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

# Hasil:
# Bagian bilangan riil:  6.0
# Bagian imajiner dari:  4.0
```

Python juga menggunakan notasi khusus untuk menandai basis dari suatu bilangan. Pada umumnya, terdapat beberapa basis bilang:

1. Basis 2 (biner)
2. Basis 8 (oktal)
3. Basis 10 (desimal)

4. Basis 16 (heksadesimal)

src/02-02/numeric_base.py

```
## Base 2 - Biner

# print(0b12)
# error:
# print(0b12)
#           ^
# SyntaxError: invalid syntax

print(0b1101)
# Hasil:
# 13
# (1101) basis biner = (1 x 2^3) + (1 x 2^2( + (0 x 2^1) + (1 x 2^0) =
(13) basis 10

## Base 8 - Octal

print(0o10) #  0 dan huruf o
# Hasil:
# 8
# (10) basis 8 = (1 x 8^1) + (0 x 8^0) = (8) basis 10

## Base 16 - Hexadecimal

print(0x1AB)
# Hasil:
# 427
# (1AB) basis 16 = (1 x 16^2) + (10 x 16^1) + (10 x 16^0) = (427) basis 10

print(type(0b1101))
print(type(0o10))
print(type(0x1AB))

# Hasil: semua akan dianggap Integer
# <type 'int'>
# <type 'int'>
# <type 'int'>
```

5.3.2. 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])
```

5.3.3. 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
print (a | b) # jika bit ada di salah satu atau kedua operand,
              # diaktifkan di hasil:
              # 0000 0111 = 7
# XOR
print (a ^ b) # jika bit ada di salah satu operand tapi tdk di keduanya,
              # diaktifkan di hasil:
              # 0000 0101 = 5
# Negasi / Not
print (-a)
print('Operator Penugasan / Assignment')
x = 50
print(x) # 50
x+=5
print(x) # x = x lama + 5 = 50 + 5 =  55
x-=2
print(x) # x = x lama - 2 = 55 - 2 = 53
x*=2
print(x) # x = x lama * 2 = 53 * 2 = 106
x/=2
print(x) # x = x lama / 2 = 106 / 2 = 53
x%=3
print(x) # x = x lama modulo 3 = 53 modulo 3 = 2.0
         # (karena pembagian terakhir berhenti di 51)
x = 55
x//=2
print(x) # x = x lama / 2, hasil dibulatkan ke bawah = 27.5
         # dibulatkan 27
x**=2
print(x) # x = x lama pangkat 2 = 27 pangkat 2 = 729
x = 7
x&=2
print(x) # x = x lama AND 2 = 7 and 2
         # 7 = 0000 0111
         # 2 = 0000 0010
         # bit hidup jika di kedua operand hidup
         # 0000 0010 = 2
x = 7
x|=2
print(x) # x = x lama OR 2 = 7 or 2
         # 7 = 0000 0111
         # 2 = 0000 0010
         # 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)
```

5.4. 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.

5.5. 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.

Chapter 6. Perintah Dasar Python

Suatu bahasa pemrograman mempunyai perintah serta fasilitas *builtin* untuk merangkai instruksi supaya bisa digunakan untuk memecahkan masalah pemrograman. Bisa dikatakan bahwa kode sumber Python yang dibuat nantinya akan terdiri atas:

1. Data
2. Fasilitas pustaka *builtin*
3. Perintah-perintah Python
4. Fasilitas pustaka pihak ketiga (jika diperlukan).

Data telah dibahas pada bab tentang "Dasar-dasar Python", khususnya tentang tipe data dasar. Tipe data yang lebih kompleks akan dibahas pada bab mendatang. Fasilitas pustaka *builtin* maupun pihak ketiga juga akan dibahas pada bab berikutnya. Untuk pembuatan program sederhana / belum kompleks, diperlukan pemahaman tentan tipe data dasar dan perintah-perintah Python (serta beberapa pustaka *builtin* dasar). Bagian ini akan membahas perinta-perintah Python.

Perintah-perintah Python - sering disebut juga dengan *statement* - terdiri atas perintah **sederhana** (*simple statement*) dan perintah **gabungan** (*compound statement*).

6.1. *Simple Statement*

Perintah sederhana / *simple statement* adalah perintah yang terdiri atas satu baris saja ataupun satu baris yang terdiri atas beberapa rangkaian perintiah kecil. Perintah untuk mengisi variabel adalah salah satu contoh dari perintah sederhana. Beberapa perintah lain:

1. Mengisi variabel
2. Ekspresi
3. Perintah *assert*
4. Perintah *pass*
5. Perintah *del*
6. Perintah *return*
7. Perintah *yield*

8. Perintah *raise*
9. Perintah *break*
10. Perintah *continue*
11. Perintah *import*
12. Perintah *global*
13. Perintah *nonlocal*

6.2. *Compound Statement*

Perintah gabungan / *compound statement* adalah perintah yang berisi sekelompok perintah lainnya.

6.2.1. 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")
```

6.2.2. while

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

```
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
```

6.2.3. for

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

```
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)
```

```python
# range(start, stop, step)
for a in range(1, 5, 2):
    print(a) # 1, 3
else:
    print("bukan selisih 2")

for a in range(20):
    if a > 0 and a % 2 == 0:
        print(a, " habis dibagi dua")
    else:
        print(a, " ganjil")
```

Chapter 7. Fungsi / *Function*

7.1. Apakah Fungsi Itu?

Fungsi (*function*) merupakan blok / unit dari program yang digunakan untuk mengerjakan suatu pekerjaan tertentu dan mengembalikan hasilnya ke pemanggil. Hasil yang dikembalikan bisa merupakan data maupun hasil eksekusi suatu *statement* untuk mengerjakan *task* tertentu. 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.

7.2. Membuat Fungsi

Rerangka fungsi di Python adalah sebagai berikut:

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

# memanggil fungsi:

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

Contoh:

src/02-04/01_function.py

```
def jumlah(arg1):   ①
    jml = 0         ②
    for a in arg1:
        jml += 1
    return jml      ③

str_obj_1 = "Zimera Corporation"  ④
str_obj_2 = "Zimera School"
str_obj_3 = "Zimera Systems"

jml_str_1 = jumlah(str_obj_1)  ⑤
jml_str_2 = jumlah(str_obj_2)  ⑥
jml_str_3 = jumlah(str_obj_3)  ⑦
```

```
print(f'String {str_obj_1} mempunyai ' + str(jml_str_1) + ' karakter')
print(f'String {str_obj_2} mempunyai ' + str(jml_str_2) + ' karakter')
print(f'String {str_obj_3} mempunyai ' + str(jml_str_3) + ' karakter')
```

① Mendefinisikan nama fungsi (**jumlah**) dan argumen dari fungsi.

② Awal dari **statement** yang akan dikerjakan jika fungsi tersebut dipanggil. Sering juga disebut dengan istilah "body of function".

③ Setelah diproses, nilai akan dikembalikan ke pemanggil menggunakan *return*.

④ Definisi variabel yang akan diproses oleh kode sumber.

⑤ Pemanggilan fungsi pertama.

⑥ Pemanggilan fungsi kedua.

⑦ Pemanggilan fungsi ketiga, Dari tiga pemanggilan ini, kita bisa melihat *reusable code*, sekali didefinisikan, setelah itu tidak perlu menulis ulang lagi, tinggal dipanggil namanya beserta dengan argumen yang diperlukan.

Hasilnya adalah sebagai berikut:

```
$ python function_01.py
String Zimera Corporation mempunyai 18 karakter
String Zimera School mempunyai 13 karakter
String Zimera Systems mempunyai 14 karakter
$
```

7.3. 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.

src/02-04/02_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 = "Zimera School"

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')
```

Hasilnya adalah sebagai berikut:

```
$ python 02_function_default.py
String Zimera School mempunyai 13 karakter
String Zimera School mempunyai 1 karakter a
$
```

7.4. 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*).

src/02-04/03_function_args_kwrgs.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)

list_01 = [3, 5, 9]
list_02 = [11, 51, 20]
```

```
# Pada bagian di bawah ini, * digunakan untuk unpack list, yaitu mengambil
# semua isi dari list.
# 3 + 5 +9 + 11 + 51 + 20 = 99
jml4 = penambah(*list_01, *list_02)
print(jml4)

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')
```

Hasilnya adalah sebagai berikut:

```
$ python 03_function_args_kwargs.py
1
68
55
99
a/b/c
a-b-c
{'product_id': 'P001', 'product_name': 'Laptop'}
Nilai product_id = P001
Nilai product_name = Laptop
$
```

Dengan adanya jumlah argumen tidak pasti serta penggunaan *args dan **kwargs, maka urutan definisi argumen dalam suatu deklarasi fungsi menjadi penting. Urutan deklarasi argumen suatu fungsi yang benar adalah sebagai berikut:

1. Argumen standar
2. *args

3. **kwargs

7.5. 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.

src/02-04/04_lambda.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))
```

Hasilnya adalah sebagai berikut:

```
$
200
180
$
```

7.6. __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()**.

src/02-04/05_main.py

```
def jumlah(arg1):
    jml = 0
    for a in arg1:
        jml += 1
    return jml
def main():
    str_obj = "Zimera Corporation"
    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()
```

Hasilnya adalah sebagai berikut:

```
$
String Zimera Corporation mempunyai 18 karakter
$
```

Chapter 8. Struktur Data di Python

Struktur data merupakan pengorganisasian, pengelolaan, serta penyimpanan data untuk akses data yang efisien. Python mempunyai beberapa model data. Pembahasan mengenai struktur data merupakan pembahasan lebih lanjut lagi setelah tipe data dasar. Pada tipe data dasar, kita sudah mengetahui bahwa ada beberapa tipe data bawaan (sering disebut dengan **primitive**) dan paling mendasar. Tipe data dasar tersebut biasanya memang hanya digunakan untuk keperluan yang sederhana dan sifatnya individual.

Tipe data primitif tersebut kemudian bisa digunakan untuk mengkonstruksi tipe data yang lebih kompleks. Secara umum, tipe data yang dibentuk dari konstruksi tipe data primitif tersebut dikenal dengan istilah *struktur data* atau tipe data *agregat*. Istilah lain yang juga cukup sering digunakan adalah *composite data type* atau *compound data type*. Saat pemrograman melakukan proses untuk mengkonstruksi *composite data type*, maka dikatakan bahwa pemrogram melakukan *composition* / komposisi.

8.1. 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.

src/02-05/01_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()
print(daftar_makanan[-3])
# bakso

print()
print(daftar_makanan[-2:])
# hasil: ['pecel', 'nila bakar']

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

daftar2 = daftar_makanan + ['oseng tempe', 'sayur pisang']
print()
print(daftar2)
# hasil: ['soto', 'bakso', 'pecel', 'nila bakar', 'oseng tempe', 'sayur
pisang']
jml_makanan = len(daftar2)
print(f'ada {jml_makanan} jumlah makanan')

# list bersifat mutable
daftar2[1] = 'mie ayam'
print()
print(daftar2)
# hasil: ['soto', 'mie ayam', 'pecel', 'nila bakar', 'oseng tempe', 'sayur
pisang']

# index 2 diganti sampai sebelum index 4
daftar2[2:4] = ['pecel lele','nila goreng']
print()
print(daftar2)
# hasil: ['soto', 'mie ayam', 'pecel lele', 'nila goreng', 'oseng tempe',
'sayur pisang']

# list bisa berada di dalam list
daftar2[1] = ['mie ayam biasa', 'mie ayam bakso', 'mie ayam istimewa']
print()
print(daftar2)
# hasil: ['soto', ['mie ayam biasa', 'mie ayam bakso', 'mie ayam
istimewa'],
#       'pecel lele', 'nila goreng', 'oseng tempe', 'sayur pisang']
print(daftar2[1])
# hasil: ['mie ayam biasa', 'mie ayam bakso', 'mie ayam istimewa']
print(daftar2[1][2])
# hasil: mie ayam istimewa
```

Hasilnya adalah sebagai berikut:

```
$ python 01_list.py
['soto', 'bakso', 'pecel', 'nila bakar']

soto
bakso
pecel
nila bakar

soto

nila bakar

bakso

['pecel', 'nila bakar']

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

['soto', 'bakso', 'pecel', 'nila bakar', 'oseng tempe', 'sayur pisang']
ada 6 jumlah makanan

['soto', 'mie ayam', 'pecel', 'nila bakar', 'oseng tempe', 'sayur pisang']

['soto', 'mie ayam', 'pecel lele', 'nila goreng', 'oseng tempe', 'sayur
pisang']

['soto', ['mie ayam biasa', 'mie ayam bakso', 'mie ayam istimewa'], 'pecel
lele', 'nila goreng', 'oseng tempe', 'sayur pisang']
['mie ayam biasa', 'mie ayam bakso', 'mie ayam istimewa']
mie ayam istimewa
$
```

Python menyediakan banyak fungsi untuk memanipulasi list, silahkan melihat selengkapnya dengan perintah help(list) dari prompt Python. Berikut adalah contoh kode sumber yang berisi berbagai fungsi / *method* untuk memanipulasi list:

src/02-05/02_list.py

```
the_list = [0, "satu", 2, "tiga", 4]
print(the_list)

the_list.append('lima')
print(the_list)

the_list_01 = [6, 'tujuh', 8]
the_list.extend(the_list_01)
print(the_list)
```

```
the_list.insert(4, 'nomor empat')
print(the_list)

the_list.remove('nomor empat')
print(the_list)

the_list_cadangan = []
the_list_cadangan.extend(the_list)
print(the_list_cadangan)
ambil_pop_terakhir = the_list.pop()
print(ambil_pop_terakhir)
print(the_list)
ambil_pop_2 = the_list.pop(2)
print(ambil_pop_2)
print(the_list)

the_list.clear()
print(the_list)

the_list = the_list_cadangan
print(the_list)

print(the_list.index('tujuh'))
print(the_list.index(4,2,6))

the_list.append(4)
print(the_list)
print(the_list.count(4))

#the_list.sort()
#print(the_list)
# TypeError: '<' not supported between instances of 'str' and 'int'
# => hanya bisa sort untuk angka

the_list.reverse()
print(the_list)

# the_list_02 sebenarnya merujuk ke the_list
the_list_02 = the_list
# the_list_shallow akan menghasilkan baru
the_list_shallow = the_list.copy()

# di bawah ini, jika yang clear adalah the_list maka the_list_02 akan ikut
# hilang, sementara the_list_shallow tidak.
print(the_list)
print(the_list_02)
the_list.clear()
print(the_list)
print(the_list_02)
print(the_list_shallow)

the_list = the_list_shallow
```

```
print(the_list)

# hapus data pada indeks ke 5 dan 6:
del the_list[5:7]
print(the_list)
```

Hasilnya adalah sebagai berikut:

```
$ python 02_list.py
[0, 'satu', 2, 'tiga', 4]
[0, 'satu', 2, 'tiga', 4, 'lima']
[0, 'satu', 2, 'tiga', 4, 'lima', 6, 'tujuh', 8]
[0, 'satu', 2, 'tiga', 'nomor empat', 4, 'lima', 6, 'tujuh', 8]
[0, 'satu', 2, 'tiga', 4, 'lima', 6, 'tujuh', 8]
[0, 'satu', 2, 'tiga', 4, 'lima', 6, 'tujuh', 8]
8
[0, 'satu', 2, 'tiga', 4, 'lima', 6, 'tujuh']
2
[0, 'satu', 'tiga', 4, 'lima', 6, 'tujuh']
[]
[0, 'satu', 2, 'tiga', 4, 'lima', 6, 'tujuh', 8]
7
4
[0, 'satu', 2, 'tiga', 4, 'lima', 6, 'tujuh', 8, 4]
2
[4, 8, 'tujuh', 6, 'lima', 4, 'tiga', 2, 'satu', 0]
[4, 8, 'tujuh', 6, 'lima', 4, 'tiga', 2, 'satu', 0]
[4, 8, 'tujuh', 6, 'lima', 4, 'tiga', 2, 'satu', 0]
[]
[]
[4, 8, 'tujuh', 6, 'lima', 4, 'tiga', 2, 'satu', 0]
[4, 8, 'tujuh', 6, 'lima', 4, 'tiga', 2, 'satu', 0]
[4, 8, 'tujuh', 6, 'lima', 2, 'satu', 0]
$
```

8.2. Tuple

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

src/02-05/03_tuple.py

```
the_data = 234, 'data 1', 'data 2', 343
print(the_data)
# hasil: (234, 'data 1', 'data 2', 343)

print(the_data[2])
# hasil: data2
```

```
# the_data[2] = 'change this'
# error: TypeError: 'tuple' object does not support item assignment

data2 = 'data x', 'data y', (123, 321)
print(data2)
# hasil: ('data x', 'data y', (123, 321))
print(data2[2][1])
# hasil: 321
for a in data2:
    print(a)
# hasil:
#   data x
#   data y
#   (123, 321)

# membuat tuple yang hanya berisi 1:
data3 = 435,
print(data3)
# hasil: (435,)
```

Hasilnya adalah sebagai berikut:

```
$ python 03_tuple.py
(234, 'data 1', 'data 2', 343)
data 2
('data x', 'data y', (123, 321))
321
data x
data y
(123, 321)
(435,)
$
```

Untuk mengetahui fungsi serta *method* untuk tuple, lihat juga **help(tuple)** dari REPL.

8.3. Sets

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

src/02-05/04_set.py

```
proglang = {'Rust', 'Python', 'Go', 'Rust'}
print(proglang)
tambahan = ('Ruby', 'Lua')
```

```
proglang.add(tambahan)
print(proglang)
print('Rust' in proglang)

huruf = set('Zimera Corp.')
print(huruf)

huruf2 = set()
print(huruf2)
huruf2.add('Zimera Corp.')
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)
```

Hasilnya adalah sebagai berikut:

```
$
{'Go', 'Rust', 'Python'}
{('Ruby', 'Lua'), 'Go', 'Rust', 'Python'}
True
{'o', 'p', 'i', '.', 'a', 'Z', ' ', 'C', 'r', 'm', 'e'}
set()
{'Zimera Corp.'}
{'o', 's', 'i', 'a', 'n', 'd', 'e', 'I'}
{'M', 'a', 'k', 'd', 'r', 'e'}
{'o', 's', 'i', 'n', 'I'}
{'o', 'M', 's', 'i', 'a', 'n', 'k', 'd', 'r', 'e', 'I'}
{'e', 'a', 'd'}
{'s', 'k', 'r', 'o', 'M', 'i', 'n', 'I'}
$
```

Lihat juga **help(set)** dari REPL.

8.4. Dictionary

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

src/02-05/05_dict.py

```
rumah = {'H-304': 'Bambang Purnomosidi', 'H-303': 'Zaky Aditya'}
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))
```

Hasilnya adalah sebagai berikut:

```
$
{'H-304': 'Bambang Purnomosidi', 'H-303': 'Zaky Aditya'}
dict_items([('H-304', 'Bambang Purnomosidi'), ('H-303', 'Zaky Aditya')])
Bambang Purnomosidi
Rumah nomor H-304 adalah tempat tinggal keluarga Bambang Purnomosidi
Rumah nomor H-303 adalah tempat tinggal keluarga Zaky Aditya
True
False
False
['H-303', 'H-304']
$
```

Lihat juga **help(dict)** dari REPL

Chapter 9. Modul dan Conda

9.1. 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. Istilah lain untuk modul standar ini adalah **pustaka standar**, **standard library**, **builtin functions**, dan **fungsi bawaan Python**. Saat melakukan instalasi interpreter Python, modul standar ini sudah bisa langsung digunakan dengan setelah sebelumnya mendekralasikan penggunakan modul tersebut menggunakan perintah **import**.

src/02-06/01-modul-standar.py

```
import os
import platform

print(os.name)
print(platform.system())
print(platform.release())
print(os.getcwd())
```

Jika dijalankan dari command line / shell:

```
$ python 01-modul-standar.py
posix
Linux
5.18.0-4-amd64
/home/bpdp/kerjaan/git-repos/oldstager/current/github/zimera-
school/books/pemrograman-python/src/02-06
$
```

9.2. 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.

src/02-06/modul02.py

```
# modul02.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 modul02.py
87
$
```

Jika di-import:

```
>>> import modul02
>>> modul02.penambah(12,23,12,23)
70
>>>
```

Saat menemui perintah **import modul02**, python akan mencari isi dari modul standar. Jika tidak ada, maka akan dicari modul02.py pada:

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

```
$ echo $PYTHONPATH

$ python
Python 3.10.6 (main, Oct 24 2022, 16:07:47) [GCC 11.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import sys
>>> print(sys.path)
['', '/home/bpdp/software/python-dev-tools/miniconda39/envs/py310-
```

```
utdi/lib/python310.zip', '/home/bpdp/software/python-dev-
tools/miniconda39/envs/py310-utdi/lib/python3.10',
'/home/bpdp/software/python-dev-tools/miniconda39/envs/py310-
utdi/lib/python3.10/lib-dynload', '/home/bpdp/software/python-dev-
tools/miniconda39/envs/py310-utdi/lib/python3.10/site-packages']
>>>
```

9.3. 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
------------------ ---------
aiohttp            3.8.1
aiosignal          1.2.0
anyio              3.5.0
ariadne            0.16.1
async-timeout      4.0.2
attrs              21.4.0
backoff            2.2.1
certifi            2022.9.24
charset-normalizer 2.0.4
click              8.0.4
frozenlist         1.2.0
gql                3.4.0
graphql-core       3.2.3
grpcio             1.50.0
grpcio-tools       1.50.0
h11                0.12.0
idna               3.4
multidict          6.0.2
pip                22.2.2
protobuf           4.21.9
setuptools         65.5.0
six                1.16.0
sniffio            1.2.0
starlette          0.19.1
typing_extensions  4.3.0
uvicorn            0.19.0
wheel              0.37.1
yarl               1.8.1
$
```

Untuk menginstall paket, gunakan:

```
$ pip install jupyterlab
Collecting jupyterlab
  Downloading jupyterlab-3.5.0-py3-none-any.whl (8.8 MB)

━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━
━━━━━━ 8.8/8.8 MB 4.4 MB/s eta 0:00:00
Collecting jinja2>=2.1
  Using cached Jinja2-3.1.2-py3-none-any.whl (133 kB)
Collecting jupyter-server<3,>=1.16.0
  Downloading jupyter_server-1.23.2-py3-none-any.whl (346 kB)

━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━
━━━━━━ 346.4/346.4 kB 1.9 MB/s eta 0:00:00
Collecting jupyter-core
  Downloading jupyter_core-5.0.0-py3-none-any.whl (91 kB)

━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━
━━━━━━ 91.1/91.1 kB 710.6 kB/s eta 0:00:00
Collecting packaging
  Using cached packaging-21.3-py3-none-any.whl (40 kB)
Collecting ipython
  Downloading ipython-8.6.0-py3-none-any.whl (761 kB)

━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━
━━━━━━ 761.1/761.1 kB 3.2 MB/s eta 0:00:00
...
...
...
Collecting psutil
  Downloading psutil-5.9.4-cp36-abi3-
manylinux_2_12_x86_64.manylinux2010_x86_64.manylinux_2_17_x86_64.manylinux
2014_x86_64.whl (280 kB)

━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━
━━━━━━ 280.2/280.2 kB 270.9 kB/s eta 0:00:00
Requirement already satisfied: certifi>=2017.4.17 in
/home/bpdp/software/python-dev-tools/miniconda39/envs/py310-
utdi/lib/python3.10/site-packages (from requests->jupyterlab-server~=2.10-
>jupyterlab) (2022.9.24)
Collecting urllib3<1.27,>=1.21.1
  Using cached urllib3-1.26.12-py2.py3-none-any.whl (140 kB)
Requirement already satisfied: charset-normalizer<3,>=2 in
/home/bpdp/software/python-dev-tools/miniconda39/envs/py310-
utdi/lib/python3.10/site-packages (from requests->jupyterlab-server~=2.10-
>jupyterlab) (2.0.4)
Collecting pure-eval
  Using cached pure_eval-0.2.2-py3-none-any.whl (11 kB)
Collecting asttokens>=2.1.0
  Downloading asttokens-2.1.0-py2.py3-none-any.whl (26 kB)
Collecting executing>=1.2.0
  Downloading executing-1.2.0-py2.py3-none-any.whl (24 kB)
```

```
Requirement already satisfied: six in /home/bpdp/software/python-dev-
tools/miniconda39/envs/py310-utdi/lib/python3.10/site-packages (from
asttokens>=2.1.0->stack-data->ipython->jupyterlab) (1.16.0)
Collecting cffi>=1.0.1
  Using cached cffi-1.15.1-cp310-cp310-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl (441 kB)
Collecting soupsieve>1.2
  Using cached soupsieve-2.3.2.post1-py3-none-any.whl (37 kB)
Collecting webencodings
  Using cached webencodings-0.5.1-py2.py3-none-any.whl (11 kB)
Collecting pycparser
  Using cached pycparser-2.21-py2.py3-none-any.whl (118 kB)
Installing collected packages: webencodings, wcwidth, Send2Trash, pytz,
pure-eval, ptyprocess, pickleshare, mistune, json5, ipython-genutils,
fastjsonschema, executing, backcall, websocket-client, urllib3, traitlets,
tornado, tomli, tinycss2, soupsieve, pyzmq, python-dateutil, pyrsistent,
pyparsing, pygments, pycparser, psutil, prompt-toolkit, prometheus-client,
platformdirs, pexpect, parso, pandocfilters, nest-asyncio, MarkupSafe,
jupyterlab-pygments, entrypoints, defusedxml, decorator, debugpy, bleach,
babel, asttokens, terminado, stack-data, requests, packaging, matplotlib-
inline, jupyter-core, jsonschema, jinja2, jedi, cffi, beautifulsoup4,
nbformat, jupyter-client, ipython, argon2-cffi-bindings, nbclient,
ipykernel, argon2-cffi, nbconvert, jupyter-server, notebook-shim,
jupyterlab-server, nbclassic, notebook, jupyterlab
Successfully installed MarkupSafe-2.1.1 Send2Trash-1.8.0 argon2-cffi-
21.3.0 argon2-cffi-bindings-21.2.0 asttokens-2.1.0 babel-2.11.0 backcall-
0.2.0 beautifulsoup4-4.11.1 bleach-5.0.1 cffi-1.15.1 debugpy-1.6.3
decorator-5.1.1 defusedxml-0.7.1 entrypoints-0.4 executing-1.2.0
fastjsonschema-2.16.2 ipykernel-6.17.1 ipython-8.6.0 ipython-genutils-
0.2.0 jedi-0.18.1 jinja2-3.1.2 json5-0.9.10 jsonschema-4.17.0 jupyter-
client-7.4.7 jupyter-core-5.0.0 jupyter-server-1.23.2 jupyterlab-3.5.0
jupyterlab-pygments-0.2.2 jupyterlab-server-2.16.3 matplotlib-inline-0.1.6
mistune-2.0.4 nbclassic-0.4.8 nbclient-0.7.0 nbconvert-7.2.5 nbformat-
5.7.0 nest-asyncio-1.5.6 notebook-6.5.2 notebook-shim-0.2.2 packaging-21.3
pandocfilters-1.5.0 parso-0.8.3 pexpect-4.8.0 pickleshare-0.7.5
platformdirs-2.5.4 prometheus-client-0.15.0 prompt-toolkit-3.0.32 psutil-
5.9.4 ptyprocess-0.7.0 pure-eval-0.2.2 pycparser-2.21 pygments-2.13.0
pyparsing-3.0.9 pyrsistent-0.19.2 python-dateutil-2.8.2 pytz-2022.6 pyzmq-
24.0.1 requests-2.28.1 soupsieve-2.3.2.post1 stack-data-0.6.1 terminado-
0.17.0 tinycss2-1.2.1 tomli-2.0.1 tornado-6.2 traitlets-5.5.0 urllib3-
1.26.12 wcwidth-0.2.5 webencodings-0.5.1 websocket-client-1.4.2
$
```

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

9.4. 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 ①
conda install ②
conda remove ③
conda update ④
conda update --all ⑤
```

① daftar paket yang sudah terinstall

② install paket x

③ uninstall paket x

④ update paket x

⑤ 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 create -n py311-utdi python=3.11 ①
conda env list ②
conda activate nama-env ③
conda create -p /home/bpdp/py36 python=3.6 ④
```

① membuat env dengan nama py311-utdi

② melihat env apa saja yang ada

③ mengaktifkan environment

④ membuat environment pada direktori tertentu, dengan versi Python tertentu

Chapter 10. Operasi I/O

Istilah Operasi I/O (sering juga disebut **input/output**, **IO**, dan **io**) adalah operasi masukan dan keluaran yang dilakukan terhadap peranti pemrosesan. Sebagai contoh, peranti pemrosesan bisa mendapatkan masukan data dari basis data, dari *keyboard*, atau bisa juga mendapatkan masukan data dari suatu file .txt, .csv, .json, dan lain-lain. Keluaran bisa saja ke layar, atau ke media pencetak (*printer*), atau suara, dan lain-lain. Semua bahasa pemrograman mempunyai fasilitas untuk operasi I/O, setidaknya untuk hal-hal yang cukup umum (masukan dari *keyboard*, file .txt, serta keluaran ke layar, *printer*, dan lain-lain). Jika tidak ada fasilitas, maka biasanya diperlukan **driver** untuk keperluan membaca masukan (contoh: membaca data dari basis data PostgreSQL di Python memerlukan driver **psycopg**).

10.1. Input dari Keyboard

Untuk menerima input dari keyboard, digunakan **input**.

src/02-07/input_keyboard.py

```
# 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))
```

Hasil:

```
$ python input_keyboard.py
Masukkan nama = Zaky
Umur = 20
Nama = Zaky, usia = 20
Nama = Zaky, usia =   20
$
```

10.2. Output ke Layar

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

10.3. Operasi File

Secara umum, operasi file dilakukan dengan membuka file dan kemudian melakukan proses sesuai dengan algoritma yang dikehendaki. Setiap file yang dibuka harus dibuka dengan mode tertentu supaya sesuai dengan tujuan pembukaan file (misalnya hanya untuk dibaca saja, atau bisa ditulisi, dan lain-lain). Secara lengkap, ada beberapa mode pembukaan file:

Mode	**Arti**
r	Dibuka untuk dibaca (default)
w	Dibuka untuk ditulisi (jika sudah ada, dihapus dulu), jika belum ada, dibuat.
x	Dibuka untuk dibuat baru, jika sudah ada maka gagal.
a	Dibuka untuk ditambahi isinya. Jika belum ada, dibuat.
t	Dibuka sebagai file mode teks.
b	Dibuka sebagai file mode biner.
+	Dibuka untuk ditulisi dan dibaca.

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(). Perhatikan contoh berikut (file `angka.txt` diperlukan):

src/02-07/angka.txt

```
5
3
10
21
33
6
7
```

Kode sumber Python untuk operasi file tersebut:

src/02-07/open_file_with.py

```
# 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='')
```

Hasil:

```
$ python open_file_with.py
5
3
10
21
33
6
```

```
7

True
5
3
10
21
33
6
7

sekarang hanya berisi 1 baris: tambahan
1
2
3
4
5
6
7
8
9
10
$
```

Chapter 11. 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:

```
>>> 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' ①
>>> f = open('abca.txt','f')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid mode: 'f' ②
>>> print 'abcdefg'
  File "<stdin>", line 1
    print 'abcdefg'
                 ^
SyntaxError: Missing parentheses in call to 'print'. Did you mean print
('abcdefg')? ③
>>>
```

① Kesalahan yang muncul karena file yang akan dibuka tidak ada.

② Kesalahan yang muncul karena mode pembukaan file yang salah.

③ Kesalahan yang muncul karena kesalahan sintaksis.

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 `try ⋯ except ⋯ else ⋯ finally`.

```
# 02-except.py

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'>
```

Hasil dari kode sumber tersebut jika ada kesalahan input:

```
$ python 02-except.py
masukkan angka float: r
harus memasukkan nilai float
masukkan angka: g
Error: <class 'ValueError'>
Selesai
$
```

Hasil dari kode sumber tersebut jika tidak ada kesalahan input:

```
$ python 02-except.py
masukkan angka float: 4.5
masukkan angka: 6
masukkan angka pembagi: 2
Hasil bagi:  3.0
Selesai
$
```

Chapter 12. 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.

```
# kelas.py

# definisi kelas paling sederhana
# bisa ditambah properties
class Dosen:
    pass

bpdp = Dosen()
bpdp.nama = 'Bambang Purnomosidi'

print(bpdp)
print(bpdp.nama)

class DosenSTMIKAkakom(Dosen):

    institusi = 'STMIK AKAKOM'

    # 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)
```

Chapter 13. 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.

13.1. 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

a = 200

def no_change_state():

    # jangan mengakses variable dari luar scope def func ini
    #
```

```
    return 10*10

print(no_change_state())
print(no_change_state())
print(no_change_state())
print(no_change_state())
print(no_change_state())
```

13.2. 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().

```
# iterator.py

daftar = [1,2,3,4,5,6]

# cara iterator
i_daftar = iter(daftar)

print(i_daftar)

a = 1

while a < len(daftar):
    print(next(i_daftar))
    a += 1

# cara mudah
for z in daftar:
    print(z)
```

13.3. 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)
```

13.4. 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)
```

13.5. 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
```

13.6. 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

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

13.7. Higher Order Function

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

```
# 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())
```

13.8. Closure

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

```
# 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
```

Chapter 14. 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())
```