LOGIKA ALGORITMA PEMROGRAMAN DENGAN PYTHON

Achmad Sidik, M.Kom
Arif Nurrochman, M.Kom
Muchamad Iqbal, M.Kom
Triono, M.Kom
M. Bucci Ryando, M.Kom

PT. PENA PERSADA KERTA UTAMA

LOGIKA ALGORITMA PEMROGRAMAN DENGAN PYTHON

Penulis:
Achmad Sidik, M.Kom
Arif Nurrochman, M.Kom
Muchamad Iqbal, M.Kom
Triono, M.Kom
M. Bucci Ryando, M.Kom

ISBN: 978-623-167-470-8

Design Cover:
Yanu Fariska Dewi

Layout:
Eka Safitry

PT. Pena Persada Kerta Utama
Redaksi:
Jl. Gerilya No. 292 Purwokerto Selatan, Kab. Banyumas
Jawa Tengah.
Email: penerbit.penapersada@gmail.com
Website: penapersada.id. Phone: (0281) 7771388

Anggota IKAPI: 178/JTE/2019

All right reserved
Cetakan pertama: 2024

Hak cipta dilindungi oleh undang-undang. Dilarang memperbanyak karya tulis ini dalam bentuk dan cara apapun tanpa izin penerbit

KATA PENGANTAR

Segala puji Penulis panjatkan kehadirat Mu Ya Allah atas segala Rahmat, karunia dan kesempatan yang telah diberikan untuk dapat mewujudkan cita cita menulis buku **LOGIKA ALGORITMA PEMROGRAMAN DENGAN PYTHON** buku ini berisi tentang *Teori dan Praktek Belajar Python*, buku ini juga di dilengkapi dengan contoh contoh sederhana dan cocok bagi pemula, yang akan belajar bahasa *Python*, didesign untuk Dosen, Mahasiswa, dan Umum, terutama bagi Pemula.

Buku ini tidak akan terwujud tanpa dukungan dan bantuan dari berbagai pihak. Pada kesempatan ini, tidak lupa mengucapkan terima kasih yang sebesar besarnya bagi akademisi dan praktisi dimana buku buku nya referensi penulis dalam menyusun buku ini. Pihak penerbit, keluarga, teman teman propesi serta berbagai pihak yang tidak bisa disebutkan semuanya, yang sudah membantu mendukung terwujudnya buku ini

Tangerang, Januari 2024
Ttd
Penulis

DAFTAR ISI

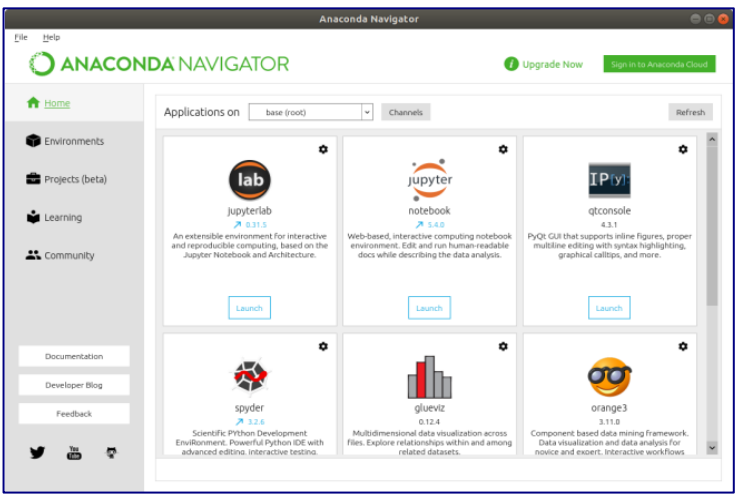
DAFTAR GAMBAR

LOGIKA ALGORITMA PEMROGRAMAN DENGAN PYTHON

BAB 1
Cara Install *Python*

Sebelum kita memulai pemrograman bahasa *Python*, terlebih dahulu dilakukan installasi *Python* komputer atau laptop yang digunakan. Setelah itu, interpreter bahasa *Python* dapat dijalankan, baik menggunakan command promt pada sistem operasi *Windows*, terminal pada sistem operasi *Linux*, atau IDE *Python* untuk mempermudah pengembangan suatu aplikasi. Berikut cara install *Python* di *Windows*, *Linux* atau *Mac OS*.

Untuk melakukan installasi *Python*, dapat menggunakan *Python* original dari situs resmi atau sumber lain. Penggunaan *Python* dari sumber lain biasa lebih mempermudah dalam penggunaan lebih lanjut, seperti penggunaan library *NumPy*, *ScriPy*, *ScyPy*, dan *matplotlib*. Pada buku ini disarankan menggunakan installasi *Python* dari sumber lain misalnya *Anaconda Navigator*. Software ini menyediakan IDE *Spyder*, *Jupyter*, serta Visual Studio Code yang dapat mempermudah menulis kode *Python*.



Gambar 1. 1 Tampilan Anaconda Navogator

A. Installasi *Python* di *Windows*

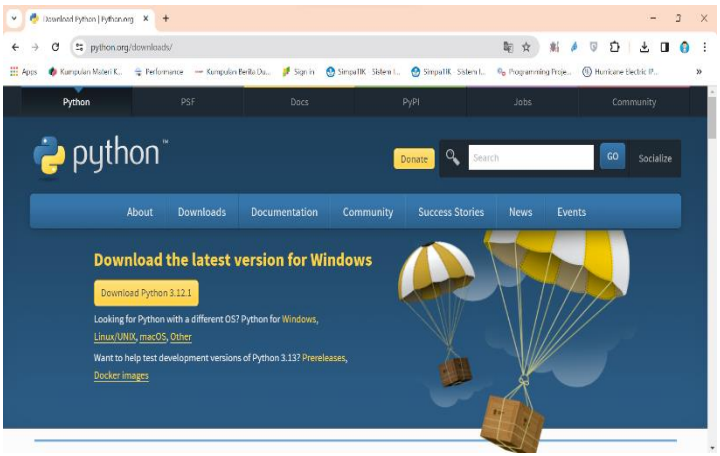
1. Download *Python* untuk *Windows*

   Kunjungi link berikut untuk mendownload *Python*, disarankan versi *Python* terbaru.
   https://www.Python.org/downloads/Windows/ atau Python v3.61 Windows.

   Instalasi *Python* di *Windows* sangat mudah. Langkah-langkanya hampir sama seperti menginstal software *Windows* pada umumnya, *next-next-finish*.

   Perhatikan ada sedikit konfigurasi yang harus dipilih ditengah tengah proses instalasi, agar perintah *Python* dapat agar dikenali di cmd atau command prompt.

   di situs resmi *Python* yang akan diinstal dalam panduan ini adalah *Python* versi 3.12.1. Download *Python*.org.

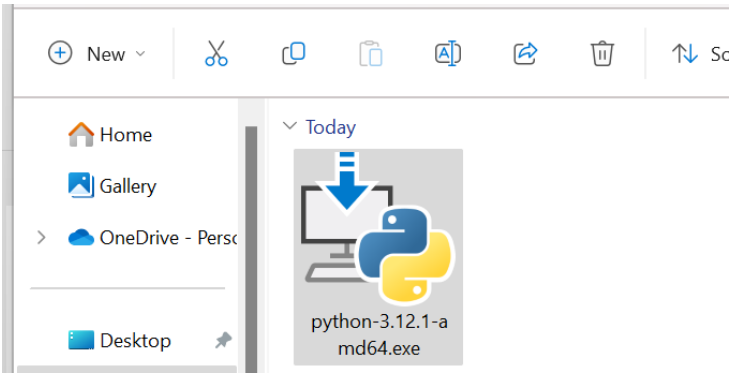


Gambar 1. 2 Tampilan *Python.org*

2. Pilih File *Python*-3.12.1.exe

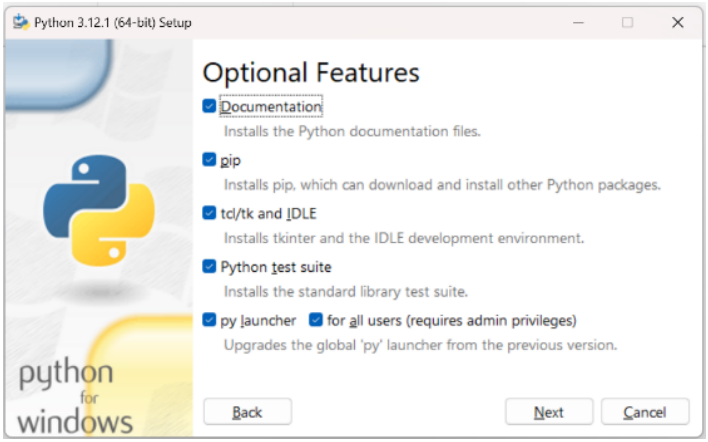
   Setelah selesai download, kita akan mendapatkan *file Python-3.12.1-amd64.exe*. *File Python-3.12.1-amd64.exe* adalah file instalator *Python*. File ini akan melakukan instalasi ke sistem *Windows*.

   Klik dua kali untuk mengeksekusinya. Seperti file gambar di bawah ini.



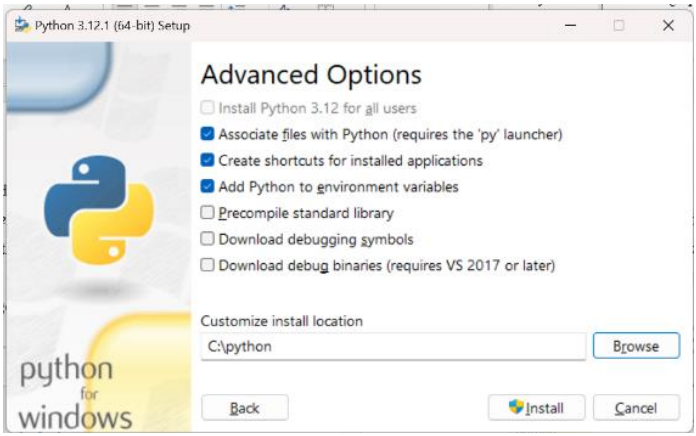
Gambar 1. 3 Tampilan *Python* 3.12.1

3. Pilih User
   Tahapan untuk memilih siapa saja yang boleh memakai *Python*.
   Pilih *Install for all users*.



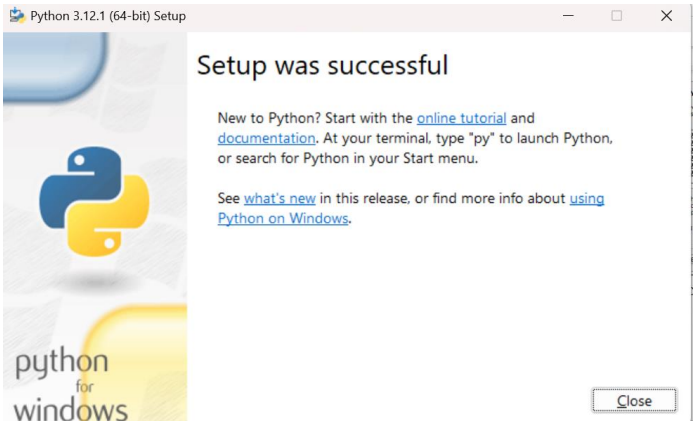
Gambar 1. 4 Tampilan select user *Python*

4. Lokasi Instalasi
   Kita menentukan lokasi *Python* akan diinstal. Contohnya kita letakkan di *C:\Python*, kemudian klik *next*.



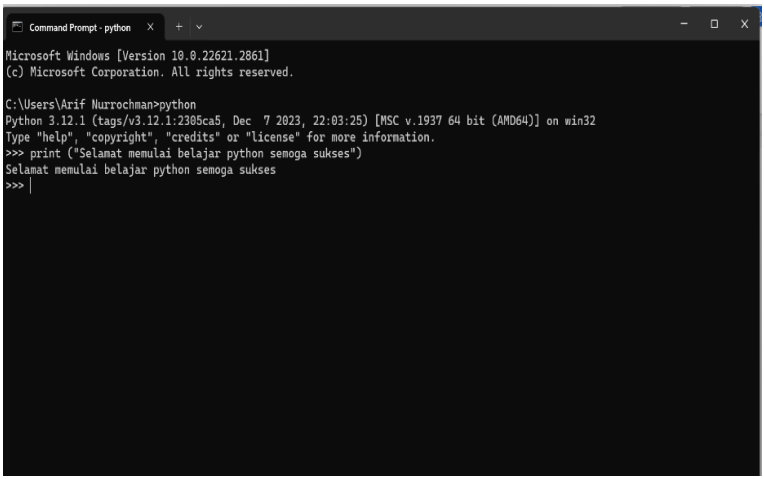
Gambar 1. 5 Pilih lokasi install *Python*

5. Finish



Gambar 1. 6 Gambar install berhasil

Kemudian coba dengan *Python* dari Cmd, ketik perintah *Python* untuk masuk ke *Python Shell* dari Cmd.



Gambar 1. 7 Tampilan *Python* interpreter

Jika tampilan seperti diatas berarti anda berhasil install *Python*.

B. Installasi *Python* di *Linux*

1. Download *Python* untuk *Linux*
   Kunjungi link berikut untuk mendownload *Python*, disarankan versi *Python* terbaru. *https://www.Python.org/downloads/source/ atau Python v3.12.1 Linux*
2. Ekstrak File Dwonload
3. Buka Direktori Folder File
4. Jalankan Melalui Terminal
   Buka terminal dengan melakukan klik kanan pada direktori folder file, pilih *Open in terminal*. Jalankan perintah dengan otoritas super user contoh diawali dengan sudo (Ubuntu) pada perintah no 1.
   1./configure script
   1 make
   1 make install

   Python Terinstall
   Langkah ini menginstal *Python* pada */usr/local/bin* dan *usr/local/lib/PythonXX* dengan *XX* adalah kode versi *Python*.

C. Installasi *Python* di Mac OS

1. Download *Python* untuk Mac OS
   Kunjungi link berikut untuk mendownload *Python*, disarankan versi *Python* terbaru. https://www.python.org/downloads/mac-osx/ atau *Python* v3.12.1 Mac OS
2. Klik 2x untuk Melakukan Installasi *Python* Lakukan installasi *Python* hingga selesai.

D. Install *Python* Anaconda Navigator

Untuk melakukan installasi *Python* menggunakan Anaconda Navigator, dapat mendownload software secara gratis pada situs resminya.

https://www.anaconda.com/download/

Download software Anaconda sesuai dengan sistem operasi yang anda gunakan. Untuk *Windows* dan Mac OS, anda dapat melakukan installasi dengan membuka langsung file hasil download di situs Anaconda Navigator.

Installasi Anaconda Navigator di *Linux*

1. Buka Direktori yang Memuat File
   Misalkan nama filenya *Anaconda3-5.1.0-Linux-x86.sh*
2. Jalankan Terminal
   Klik kanan pada direktori yang memuat file yang telah didownload, pilih *Open in terminal*. Jalankan perintah dengan otoritas super user, misalnya *sudo* untuk Ubuntu.
   1bash ./Anaconda3-5.1.0-Linux-x86.sh
3. Lakukan Proses Installasi

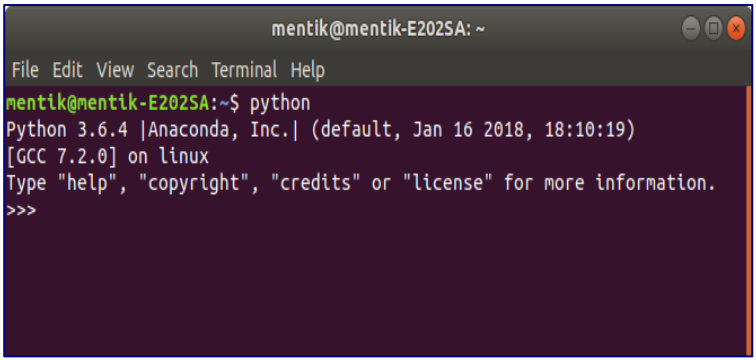
Untuk membuka Anaconda Navigator pada *Linux*, jalankan pada terminal pada direktori awal, anda dapat menuju direktori awal dengan melakukan eksekusi perintah *cd* pada terminal yang masih terbuka.

Disarankan untuk tidak menggunakan otoritas *Super User* untuk membukan Anaconda Navigator pada *Linux*.

1 anaconda-navigator

E. Menjalankan Interpreter *Python*

Untuk menguji apakah *Python* telah berhasil diinstall anda dapat menjalankan perintah berikut melalui command prompt atau terminal.



Gambar 1. 8 Tampilan *Python* interpreter *Linux*

Dengan terminal tersebut sudah dapat menjalankan interpreter *Python*.

BAB 2
Filosofi Pemrograman *Python*

A. Filosofi Pemrograman *Python*

Bahasa pemrograman *Python* ditujukan untuk menghasilkan bahasa pemrograman sederhana yang mampu memberikan fitur-fitur dari bahasa pemrograman lain termasuk *C++, C*, dan *Java*. Sehingga *Python* mampu memberikan cita rasa pemrograman yang cepat, pembelajaran yang mudah, serta mempermudah pengembangan software.

Python menjadi bahasa pemrograman open source terbesar. Setiap orang dapat berkontribusi untuk membangun program-program yang dapat digunakan oleh orang lain dalam programnya. Hal ini disebut dengan *package* atau *module.* Tercatat untuk tahun 2018, terdapat 117.181 module yang dibuat oleh seluruh developer dan dibagikan sebagai repository *Python*.

"Filosofi *Python* adalah sebagai bahasa pemrograman multipurpose, portable, object-oriented, dan high level programming yang memberikan lingkungan interaktif untuk menulis kode secara minimalis"

B. Dasar dasar pemrograman *Python*

1. Multipurpose

   Python merupakan bahasa pemrograman multiguna (*multipurpose*). Terdapat banyak modul yang mempermudah berbagai disiplin ilmu misalnya scientific computation, statistic, networking, cryptography, game development, GUI, machine learning, image processing, plotting, database, HTML/XML parsing, natural language processing, dan testing.

2. High Level Language

   Python merupakan *high level programming language* atau bahasa pemrograman tinggi. Dalam hal pemrograman secara umum, dikenal bahasa pemrograman rendah dan bahasa

pemrograman tinggi. Bahasa pemrograman rendah melakukan komunikasi langsung menggunakan bahasa mesin untuk berkomunikasi dengan hardware. Sedangkan bahasa pemrograman tinggi menggunakan library untuk melakukan komunikasi dengan hardware. Library ini berfungsi untuk melakukan terjemahan bahasa tinggi ke bahasa mesin.

3. Object Oriented Programming

   Python menggunakan fundamental pemrograman berorientasi objek atau *object oriented programming*. Bahasa pemrograman tradisional melakukan komputasi secara natural, dalam artian semua harus didefinisikan terlebih dahulu untuk dilakukan proses secara runut. Dalam perkembangan selanjutnya, dikenal *object oriented programming* (OOP). Fundamental ini membagi kode menjadi blok-blok yang dieksekusi secara paralel sebagai tread pemrosesan yang disebut dengan *object.* Dalam hal ini metode OOP akan melakukan eksekusi dengan berorientasi pada object. Object dapat berupa function dan class.

4. Interactive Environtment

   Syntax pemrograman pada *Python* merupakan turunan dari syntax yang digunakan bahasa C dan UNIX. Sebagain besar syntax *Python* mempunyai kesamaan dengan bahasa pemrograman C. Dengan mengadopsi lingkungan UNIX, bahasa pemrograman *Python* dapat diajalan dalam mode console. Ditambah dengan syntax- syntax yang dibuat untuk mempermudah pembelajaran seperti *help()* membuat *Python* menjadi bahasa pemrograman interaktif.

5. Minimalistic Design

   Struktur kode yang minimalis membuat bahasa pemrograman *Python* mempunyai tingkat keterbacaan yang jauh lebih mudah dari bahasa pemrograman lainnya. Seperti penggunaan tabulasi atau indentasi sebagai pembatas struktur. Filosofi terkait minimalisasi syntax terdapat pada dokumen

yang diterbitkan Tim Peters yang berjudul *The Zen of Python*.

6. Portability

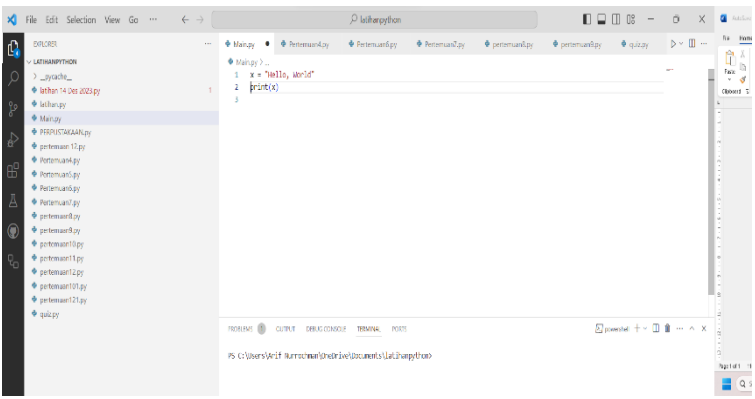
   Python didukung oleh komunitas open source-nya. Hal ini membuat *Python* dapat berjalan di berbagai platform. Termasuk *Linux*, Microsoft *Windows*, Solaris, Macintosh, dan SonyPlaystation.

7. Extensibilty

   Python mempunyai ratusan ribu modul yang dapat digunakan untuk mempermudah pengembangan suatu software atau riset. Misalnya modul untuk menghitung integral, pengguna dapat membuat program tanpa harus membuat syntax yang dapat menghitung integral.

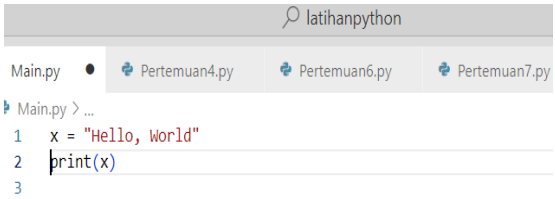
C. Memulai Pemrograman *Python*

Untuk memulai pemrograman *Python* dapat dilakukan melalui *Visual Studio Code*. Pada artikel ini digunakan *Visual Studio Code*. Syntax dapat dieksekusi melalui console (misalnya I*Python*) maupun dalam bentuk file melalui editor. File syntax *Python* disimpan dalam format (*.py).*



Gambar 2. 1 Tampilan Visual Studio Code

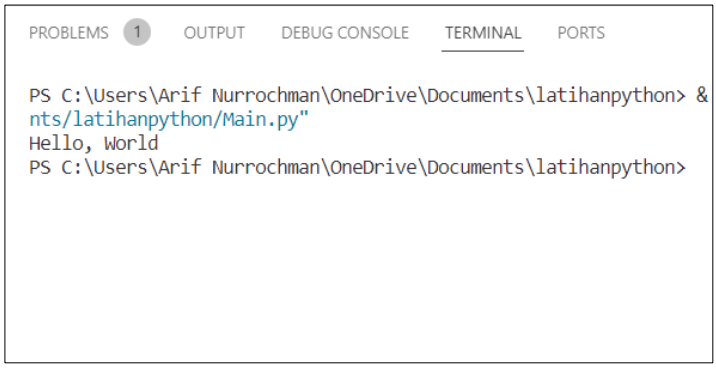
1. Buka Aplikasi yang anda gunakan
2. Membuat Program Sederhana Program Hello World print("Hello World")



Gambar 2. 2 Program Visual Studio Code

D. Simpan dan Jalankan

Misalnya save nama program dengan nama (hello_world.py). Selanjutnya program dapat dijalankan dan output dihasilkan pada console.



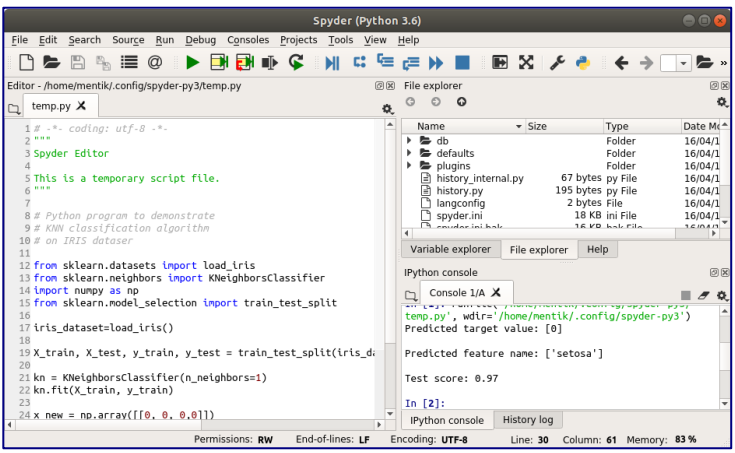
Gambar 2. 3 Program simpan dan jalankan

BAB 3
Python IDE Windows dan Linux

Sebagai bahasa pemrograman interpretatif, banyak developer yang mengembangkan *Python IDE Windows* dan *Linux* untuk memberikan layanannya. *Python* IDE atau Integrated Development Environtment adalah software yang membantu developer untuk mengembangkan aplikasi dengan menggunakan bahasa pemrograman *Python* yang dilengkapi GUI. Peggunaan *Python* IDE, dapat mempermudah kita untuk menulis kode dan menjalankan kode *Python*. Berikut artikel review *Python* IDE *Windows Linux* terbaik dan gratis.

1. *Spyder*

   Spyder adalah kependekan dari The Scientific *Python* Development EnviRonment yang merupakan IDE yang memfokuskan untuk melakukan pemrograman di ranah scientific atau keilmuan. IDE ini tersedia untuk sistem operasi *Windows*, *Linux*, dan Mac OS. *Spyder* juga didistribusikan secara langsung pada software Anaconda.

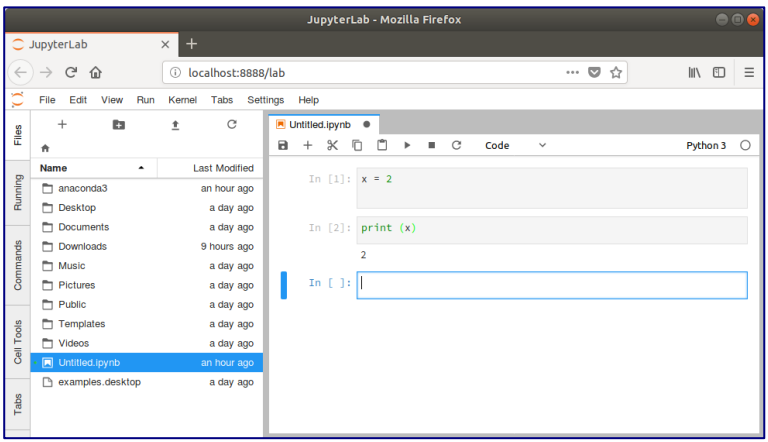


Gambar 3. 1 Tampilan program spyder

Spyder mempunyai user interface seperti text editor dengan fitur syntax highlighter untuk mempermudah keterbacaan kode. Selain itu tersedia fitur *Interactive console, Documentation viewer, Variable explorer, Find in filfes, File explorer, dan History log*. Fitur-fitur ini sangat mendukung pengguna dengan latar belakang pemrograman menggunakan MATLAB dan RStudio.
Download Spyder

2. *JupyterLab*

JupyterLab adalah IDE *Python* yang dibangun dari web server komputer pengguna dalam bentuk aplikasi berbasis web. *JupyterLab* diterbitkan sejak tahun 2014 dan telah menjadi pilihan sebagai *IDE Python* khususnya analisis data. Dengan menggunakan arsitektur berbasis *web server*, *IDE* ini menjadi sangat ringan.

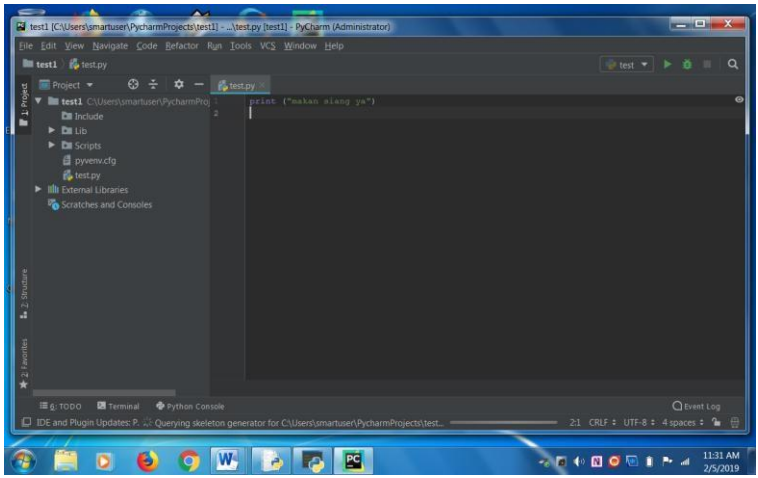


Gambar 3. 2 Tampilan program *JupyterLab*

JupyterLab memungkinkan pengguna untuk melakukan ekspor data hasil visualisasi seperti matplotlib kebentuk html dan pdf. Hal ini akan mempermudah pengguna untuk membagikan hasil analisis data, baik untuk presentasi maupun diupload ke situs web atau blog. *JupyterLab* tersedia untuk sistem operasi *Linux*, *Windows*, dan Mac OS.

3. *PyCharm*

PyCharm adalah IDE *Python* yang dikembangkan oleh JetBrains‟s. IDE ini sangat direkomendasikan bagi yang terbiasa dengan IDE bahasa pemrograman lain yang didstribusikan oleh JetBrain‟s. IDE ini tersedia gratis untuk versi komunitas. Tersedia untuk sistem operasi *Linux*, *Windows*, dan Mac OS.

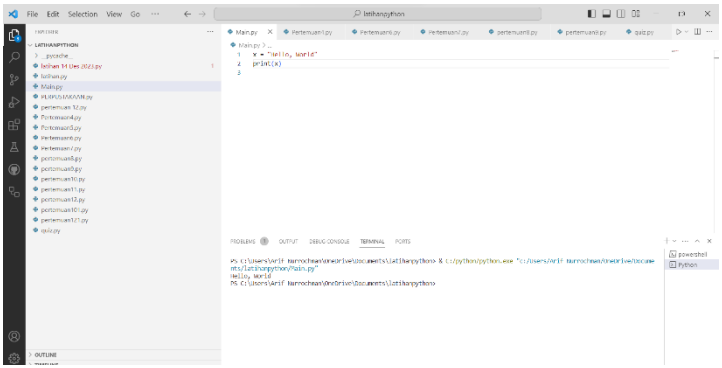


Gambar 3. 3 Tampilan program *PyCharm*

Sama dengan Spyder dan JupiterLab, *PyCharm* juga dapat diintegrasikan dengan library-library *Python* seperti NumPy, ScyPy, MatplotLib, dan lain-lain. *PyCharm* memiliki fitur-fitur pendukung seperti code editor, syntax highlighter, debugger, integrasi GIT, SVN, dan Mercurial. *Download PyCharm*

4. *Visual Studio Code*

Visual Studio Code adalah sebuah teks editor multiplatform yang komplit dan handal buatan Microsoft. Selain tersedia untuk *Windows*, Visual Studio Code juga tersedia untuk versi *Linux* dan Mac.



Gambar 3. 4 Tampilan program Visual Studio Code

Sama dengan Spyder dan JupiterLab, *Visual Studio Code* juga dapat diintegrasikan dengan library-library *Python* seperti NumPy, ScyPy, MatplotLib, dan lain-lain. *PyCharm* memiliki fitur-fitur pendukung seperti code editor, syntax highlighter, debugger, integrasi GIT, SVN, dan Mercurial. *Download PyCharm*

BAB 4
Pengertian Pemrograman *Python*

Pengertian bahasa pemrograman *Python* adalah bahasa pemrograman tingkat tinggi yang dapat melakukan eksekusi, sejumlah intruksi multiguna secara langsung (interpretatif) dengan metode orientasi objek (Object Oriented Programming) serta menggunakan semantik dinamis untuk memberikan tingkat keterbacaan syntax. Sebagai bahasa pemrograman tingkat tinggi, *Python* dapat dipelajari dengan mudah karena sudah dilengkapi dengan manajemen memori otomatis (pointer).

A. *Python* Pemrograman Open Source

Python dapat digunakan secara bebas, bahkan untuk kepentingan komersial sekalipun. Banyak perusahaan yang mengembangkan bahasa pemrograman *Python* secara komersial juga memberikan layanannya. Misalnya Anaconda Navigator, adalah salah satu aplikasi untuk pemrograman *Python* yang dilengkapi dengan tool-tool pengembangan aplikasi.

B. Rapid Application Development (RAD)

Python diklaim mampu memberikan kecepatan dan kualitas untuk membangun aplikasi bertingkat tinggi (Rapid Application Development). Hal ini didukung oleh adanya library dengan modul-modul baik standar maupun tambahan misalnya *NumPy*, *SciPy*, dan lain-lain. *Python* juga mempunyai komunitas yang besar sebagai tempat tanya jawab memecahkan masalah.

Mesin pencari Google adalah contoh nyata dari penggunaan bahasa pemrograman *Python* dalam kehidupan sehari-hari. Mesin pencari ini termasuk *Rapid Application Development*, ia tidak hanya berguna untuk mencari halaman website. Kolom pencarian Google juga dapat digunakan sebagai kalkulator, membuat grafik fungsi, memprediksi cuaca, memprediksi harga saham, terjemahan, mencari dengan gambar, menanyakan hari, pemesanan tiket pesawat, dan lain-lain.

C. *Python* Mendukung Berbagai Sistem Operasi

Syntax Python dapat dijalankan dan ditulis untuk membangun aplikasi di berbagai sistem operasi.

1. *Linux/Unix*
2. *Microsoft Windows*
3. *Mac OS*
4. *Android*
5. *Java Virtual Machine*
6. *Symbian OS*
7. *Amiga*
8. *Palm*
9. *OS/2*

D. Aplikasi Penggunaan *Python*

Python digunakan di berbagai bidang pengembangan. Berikut beberapa aplikasi penggunakan *Python* yang paling populer,

1. Website dan internet
   Bahasa pemrograman *Python* dapat digunakan sebagai server side yang diintegrasikan dengan berbagai internet protokol misalnya HTML, JSON, Email Processing, FTP, dan IMAP. Selain itu, *Python* juga mempunyai library untuk pengembangan internet.
2. Penelitian ilmiah dan Numerik
   Python dapat digunakan untuk melakukan riset ilmiah untuk mempermudah perhitungan numerik. Misalnya penerapan algoritma KNN, Naive Bayes, Decision Tree, dan lain-lain.
3. Data Science dan Big Data
   Python memungkinkan untuk melakukan analisis data dari database big data.
4. Media pembelajaran pemrograman
   Python dapat digunakan sebagai media pembelajaran di universitas. *Python* sangat mudah dan hemat untuk dipelajari sebagai Object Oriented Programming dibandingkan bahasa lainnya seperti MATLAB, C++, dan C#.

E. Graphical User Interface (GUI)

Python dapat digunakan untuk membangun interface sebuah aplikasi. Tersedia library untuk membuat GUI menggunakan *Python*, misalnya *Qt, win32extension, dan GTK+*.

1. Pengembangan Software
   Python menyediakan dukungan struktur kode untuk mempermudah pengembangan software.
2. Aplikasi bisnis
   Python juga dapat digunakan untuk membuat sistem informasi baik untuk bisnis dan instansi.

BAB 5
Pengenalan Dasar *Python*

Belajar bahasa pemrograman *Python* sangat mudah, sekarang ini sangat popular dikalangan ilmuan atau researcher, karena bahasa *Python* baik digunakan untuk pengujian Artificial intelligence. Untuk memahami bahasa *Python* ada beberapa hal yang harus diketahui.

Python merupakan bahasa pemrograman tingkat tinggi yang dibuat oleh Guido van Rossum, dia adalah programmer komputer berkewarganegaraan Belanda yang lebih di kenal sebagai pencipta dan "*Benevolent Dictator for Life*" dari bahasa pemrograman *Python* yang artinya dia hanya akan memberikan keputusan akhir jika dibutuhkan.

Python banyak digunakan untuk membuat berbagai macam program, seperti: program *CLI, Program GUI (desktop), Aplikasi Mobile, Web, IoT, Game, Program untuk Hacking*, dan sebagainya.

Python juga dikenal dengan bahasa pemrograman yang mudah dipelajari, karena struktur sintaknya rapi dan mudah dipahami. (*Python* bagus untuk pemula yang belum pernah *coding*) Berikut perbedaan bahasa *Python* , C++ dan JAVA

C++ "Hello World"

```
#include <iostream.h>
main()
{
cout << "Hello World! ";
}
return 0
```

Java "Hello World"

```
class HelloWorldApp
  {
    public static void main(String[] args)
      {
       System.out.println("Hello World!");
       }
    }
```

Python

```
print "Hello world"
```

Gambar 5. 1 Perbandingan Coding *Python* dengan Lain

Python memang sangat sederhana dibandingkan bahasa yang lainnya. Tidak perlu banyak aturan membuat program *Hello World!*. Defenisi bahasa *Python* adalah sebagai berikut:
"Python is a programming language that lets you work quickly and integrate systems more effectively".

Berikut alasan kita kenapa belajar pemrograman *Python*.
1. Cocok digunakan untuk mengolah data science
2. Mudah dipelajari
3. Banyak digunakan di perusahaan besar
4. Banyak digunakan para ilmuan dibidang computer terutama machine learning
5. Banyak digunakan para peneliti / researcher

A. Persiapan Belajar Pemrograman *Python*

Peralatan yang harus dipersiapkan untuk belajar pemrograman *Python* adalah sebagai berikut:
1. *Python*: *Interpreter* yang menerjemahkan bahasa *Python* ke bahasa mesin, sehingga program bisa dijalankan.
2. *Teks Editor/IDE*: Program yang digunakan untuk menulis kode.

Bagi pengguna *Linux*, *Python* tidak perlu diinstal. Karena Sebagian besar distro *Linux* sudah menyediakannya secara default. Untuk mengeceknya, silahkan ketik perintah `Python --version` di terminal.

```
$ python --version
```

Python 3.12.1

Bagi pengguna *Windows*, silahkan baca Cara Install *Python* di *Windows* dibab sebelumnya.

Python memiliki versi saat ini ada dua macam yaitu *Python* Versi 2 dan *Python* Versi 3. Adapun perbedaannya adalah *Python* versi 2 merupakan versi yang banyak digunakan saat ini, baik dilingkungan produksi dan pengembangan.

Sementara *Python* versi 3 adalah pengembangan lanjutan dari versi 2. *Python* 3 memiliki lebih banyak fitur dibandingkan *Python* 2.

Untuk membuka *Python* 2 kita hanya menggunakan perintah `Python` saja, sedangkan *Python* 3 menggunakan perintah `Python3`.

```
-$ python -version
```

Pyhton 3.12.1

```
-$ python3 -version
```

Pyhton 3.12.1

Siapkan Teks Editor/IDE untuk Menulis Kode, Teks editor yang digunakan untuk menulis program *Python* bisa apa saja. Bahkan Notepad pun bisa.

Pada *Linux*, banyak sekali pilihan teks editor yang bisa digunakan.

Untuk teks editor berikut referesi untuk dibaca:

1. 6 Teks Editor Berbasis Teks (CLI) di *Linux* untuk Menulis Kode
2. [Review] Text Editor Visual Studio Code di *Linux*

Selain teks editor, kita juga bisa menggunakan IDE *(Integereted Development Environtment)*. Namun, nanti kita akan bahas belakangan.

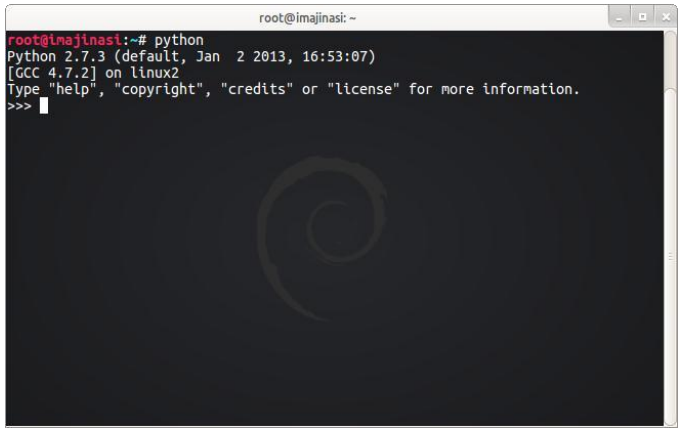
Untuk saat ini kita pakai teks editor saja dulu, biar lebih paham konsep pemrograman.

B. Mengenal Mode Interaktif *Python*

Mode interaktif merupakan fasilitas/fitur yang disediakan oleh *Python* sebagai tempat menulis kode secara interaktif.

Fitur ini dikenal juga dengan *Shell*, *Console*, REPL *(Read–Eval–Print Loop)*, interpreter, dan lain lain.

Cara membuka mode interaktif adalah dengan mengetik perintah *Python* pada terminal.



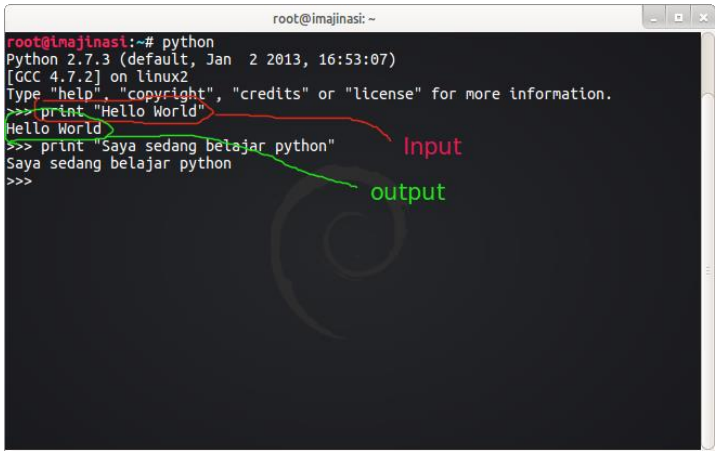
Gambar 5. 2 Tampilan Mode Interaktif *Python* di Linux

Untuk keluar dari mode interaktif tekan `Ctrl+d` atau ketik perintah `exit()`.

Tanda >>> adalah artinya *Python* siap menerima perintah. Terdapat juga tanda ... yang berarti *secondary prompt* atau *sub prompt*, biasanya muncul saat membuat blok kode dan menulis perintah tunggal dalam beberapa baris.

Mari kita coba memberikan perintah `print`, perintah ini berfungsi untuk mencetak teks ke layar.

Cobalah tulis `print "Hello World"` kemudian tekan `Enter`.



Gambar 5. 3 Tampilan untuk input dan output

Perintah yang kita tulis langsung dieksekusi dan ditampilkan hasilnya. Inilah mode interaktif, setiap kode atau perintah yang diketik akan direspon langsung oleh *Python*.

Kita bisa memanfaatkan mode interaktif ini untuk:

1. Uji coba suatu fungsi
2. Eksperimen modul tertentu
3. Kalkulator
4. Mencari bantuan tentang fungsi tertentu dll

Hal yang perlu kita coba adalah mencari bantuan tentang fungsi tertentu, karena akan membantu sekali dalam mempelajari *Python*.

Ada dua fungsi yang digunakan untuk mencari bantuan:

1. fungsi `dir()` untuk melihat fungsi apa saja yang tersedia pada sebuah modul.
2. fungsi `help()` untuk membuka dokumentasi suatu fungsi.

Sebagai contoh, kita akan coba mencari tahu tentang penggunaan modul `Math`. Pertama kita impor dulu modulnya ke mode interaktif.

```
>>> import math
```

Setelah itu kita bisa melihat fungsi apa saja yang tersedia di modul tersebut.

```
>>> dir(math)
```

['__doc__', '__name__', '__package__', 'acos', 'acosh', 'asin', 'asinh', 'atan', 'atan2', 'atanh', 'ceil', 'copysign', 'cos', 'cosh', 'degrees', 'e', 'erf', 'erfc', 'exp', 'expm1', 'fabs', 'factorial', 'floor', 'fmod', 'frexp', 'fsum', 'gamma', 'hypot', 'isinf', 'isnan', 'ldexp', 'lgamma', 'log', 'log10', 'log1p', 'modf', 'pi', 'pow', 'radians', 'sin', 'sinh', 'sqrt', 'tan', 'tanh', 'trunc']

Lalu, kita bisa cari tahu cara penggunaan fungsi-fungsi tersebut dengan *`help()`*. Misalkan ingin cari tahu cara penggunaan fungsi *`pow()`*, maka kita harus memberikan perintah `help(math.pow)`. Bantuan pada fungsi *`pow`* dalam modul

matematika :*pow(...)*

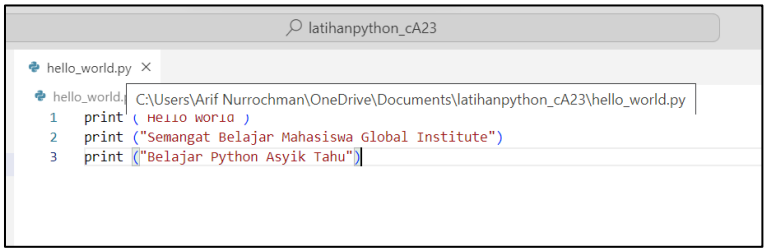
```
     pow(x, y)
     Return x**y (x to the power of y).
(END)
```

*untuk keluar dari dokumentasi tekan q

C. Menulis Skrip *Python*

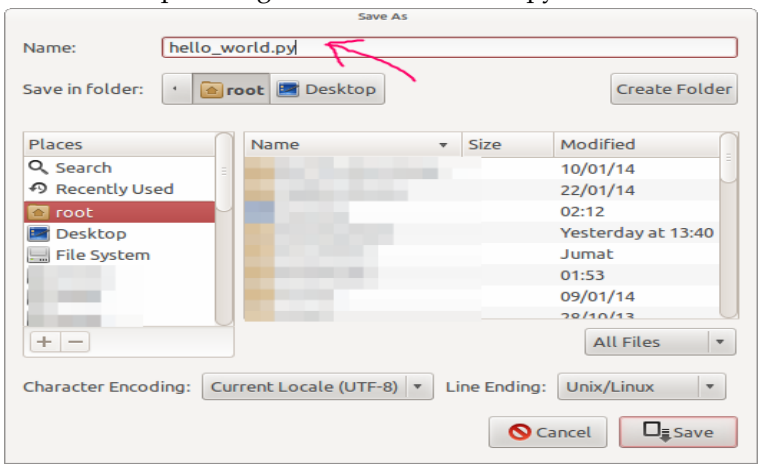
Program yang kita tulis dalam mode interaktif tidak akan disimpan. Setelah mode interaktif ditutup, program akan hilang. Oleh karena itu, kita harus membuat skrip.

Silahkan gunakan teks editor untuk menulis skrip seperti di bawah ini.



Gambar 5. 4 Tampilan teks editor menulis skrip

Setelah itu simpan dengan nama hello_world.py



Gambar 5. 5 Tampilan menyimpan dteks editor

Kemudian untuk menjalankan skripnya, gunakan perintah berikut:

```
python nama_skrip.py
```

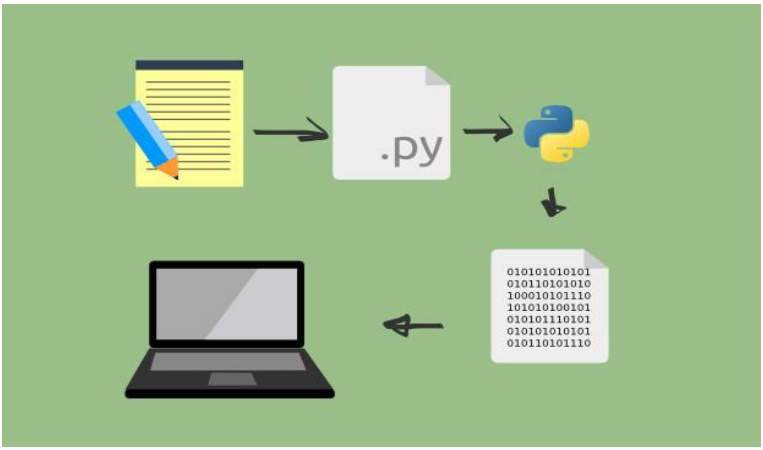
Pastikan mengetik perintah tersebut pada direktori tempat menyimpan skripnya.

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\Arif Nurrochman\OneDrive\Documents\latihanpython_cA23> & C:/python/python.exe "
python_cA23/hello_world.py"
Hello World
Semangat Belajar Mahasiswa Global Institute
Belajar Python Asyik Tahu
PS C:\Users\Arif Nurrochman\OneDrive\Documents\latihanpython_cA23>
```

Gambar 5. 6 Tampilan direktori yang disimpan

D. Alur Kerja Pembuatan Program *Python*



Gambar 5. 7 Alur Kerja Program *Python*

1. Membuat skrip *Python* dengan teks `editor`.
2. Skrip *Python* diterjemahkan ke dalam kode biner oleh `(intepreter)` *Python*, sehingga komputer dapat mengerti arti perintah tersebut.

3. Komputer mengerjakan perintah tersebut.

Sampai tahap ini, kita sudah tahu cara membuat program *Python*. Selanjutnya, kita akan membuat program *Python* menggunakan *Visual Studio Code*.

BAB 6
Penulisan Sintaks *Python*

Setelah mempersiapkan segala perlengkapan untuk *coding Python* dan mengetahui cara membuat program *Python*, harus mempelajari aturan-aturan penulisan sintaks *Python* yang harus dipatuhi.

Buku ini, akan membahas beberapa aturan dasar penulisan sintaks *Python* yang harus diketahui. Agar nanti mudah dalam menulis program.

A. Penulisan Statement *Python*

Statement adalah sebuah intruksi atau kalimat perintah yang akan dieksekusi oleh komputer.

Contoh:

```
print("Hello World!")
print("Belajar Bahasa Python dari Nol") nama =
```

Penulisan satu statement tidak diakhiri dengan tanda titik- koma. Sedangkan, bila ingin menulis lebih dari satu statement dalam satu baris, maka harus memisahnya dengan titik-koma. Contoh:

```
print("Hello");
print("World");
print("Tutorial Python untuk Pemula")
nama_depan = "belajar"; nama_belakang = "python"
```

Tapi menurut beberapa *style guide Python*, tidak dianjurkan menulis lebih dari satu statement dalam satu baris. Karena akan sulit dibaca.

B. Penulisan String pada *Python*

String adalah teks atau kumpulan dari karakter. String dalam pemrograman biasanya ditulis dengan dibungkus menggunakan tanda petik. Bisa menggunakan tanda petik tunggal maupun ganda.

Contoh:

```
judul = "Belajar Pemrograman Python dari Nol"
penulis = 'Global Institute'
```

Atau bisa menggunakan triple tanda petik.

Contoh:

```
judul = """ Belajar Pemrograman Python dari Nol """
penulis = ''' Global Institute '''
```

C. Penulisan Case pada *Python*

Sintak *Python* bersifat *case sensitive,* artinya teksini dengan TeksInidibedakan.

Contoh:

```
judul = "Belajar Dasa-dasar Python"
Judul = "Belajar Membuat Program Python"
```

Antara variabel jududldengan Judulitu dibedakan…

`Case Style`

Menurut rekomendasi *style guide* Google, berikut ini contoh penulisan *case* yang disarankan:

Snake Case digunakan pada:

```
module_name, package_name, method_name,
function_name, , global_var_name,
instance_var_name, function_parameter_name,
local_var_name.
```

CamelCase digunakan Pada:

```
ClassName, ExceptionName
```

ALL CAPS digunakan Pada:

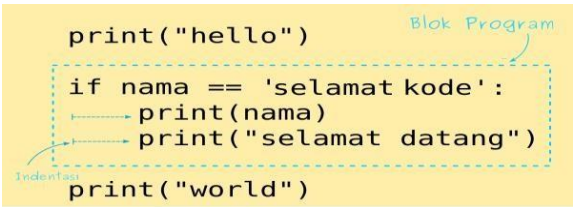
```
GLOBAL_CONSTANT_NAME
```

Silakan dibaca ada 4 Macam Gaya Penulisan Case dalam Pemrograman.

D. Penulisan Blok Program *Python*

Blok program adalah kumpulan dari beberapa statement yang digabungkan dalam satu blok.

Penulisan blok program harus ditambahkan indentasi (tab atau spasi 2x/4x).



Gambar 6. 1 Tampilan Blok Program

Contoh yang benar:

blok percabangan if

```
if username == 'belajarpython':
print("Selamat Datang Admin")
print("Silahkan ambil tempat duduk")
```

blok percabangan for

```
for i in range(10): print(i)
```

Contoh yang salah:

blok percabangan if

```
if username == global institute':
print("Selamat Datang Admin")
print("Silahkan ambil tempat duduk")
```

blok percabangan for

```
for i in range(10):
print i
```

Ada beberapa macam blok program yang digunakan pada bahasa *Python*:

1. Blok Percabangan
2. Blok Perulangan
3. Blok Fungsi
4. Blok Class
5. Blok Exception
6. Blok With

E. Cara Penulisan Komentar pada *Python*

Komentar merupakan baris kode yang tidak akan dieksekusi. Komentar digunakan untuk memberikan informasi tambahan dan untuk menonaktifkan kode.

Ada beberapa cara menulis komentar pada pemrograman *Python*. Menggunakan Tanda Pagar (#)

Contohnya:

```
# ini adalah komentar #
Ini juga komentar
```

Menggunakan Tanda Petik

Selain untuk mengapit teks (*string*), tanda petik juga dapat digunakan untuk membuat komentar.

Contoh:

```
"Ini adalah komentar dengan tanda petik ganda"
'Ini juga komentar, tapi dengan tanda petik tunggal'
```

Penulisan komentar dengan tanda petik jarang digunakan, kebanyakan orang lebih memilih untuk menggunakan tanda pagar. Jadi tidak direkomendasikan, menggunakan Triple Tanda Petik, Sedangkan triple tanda petik, sering digunakan untuk menuliskan dokumentasi.

Contohnya: class Pagar:

```
"""Ini hanyalah contoh komentar"""
    def __init__(self, warna, tinggi, bahan):
    self.warna = warna
    self.tinggi = tinggi
    self.bahan = bahan
```

Hasilnya:

```
$ python kelas_pagar.py
```

kelas pagar untuk membuat objek pagar. dibuat oleh Belajar *Python* sebagai contoh saja.

tekan [enter] untuk melihat bantuan (dokumentasi) kelas: Setelah *Enter* ditekan

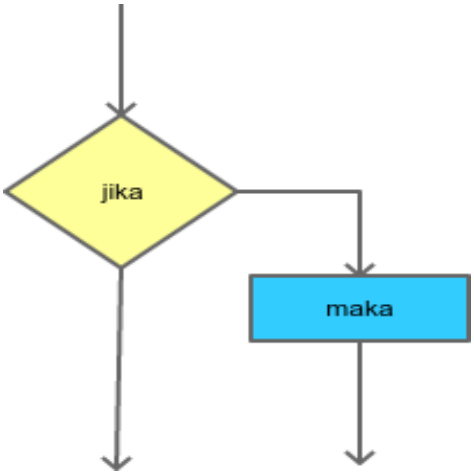
Help on class Pagar in module main:

```
class Pagar
| kelas pagar untuk membuat objek pagar.
| dibuat oleh Belajar Python
| sebagai contoh saja.
|
| Methods defined here:
|
| __init__(self, warna, tinggi, bahan) (END)
```

BAB 7
Percabangan dan Perulangan

Apa itu percabangan dan kenapa dinamakan percabangan karena dalam algoritma dan flowchart, Istilah ini sebenarnya untuk menggambarkan alur program yang bercabang.

Pada flow chart, logika "*jika…maka*" digambarkan dalam bentuk cabang.



Gambar 7. 1 Tampilan Percabangan

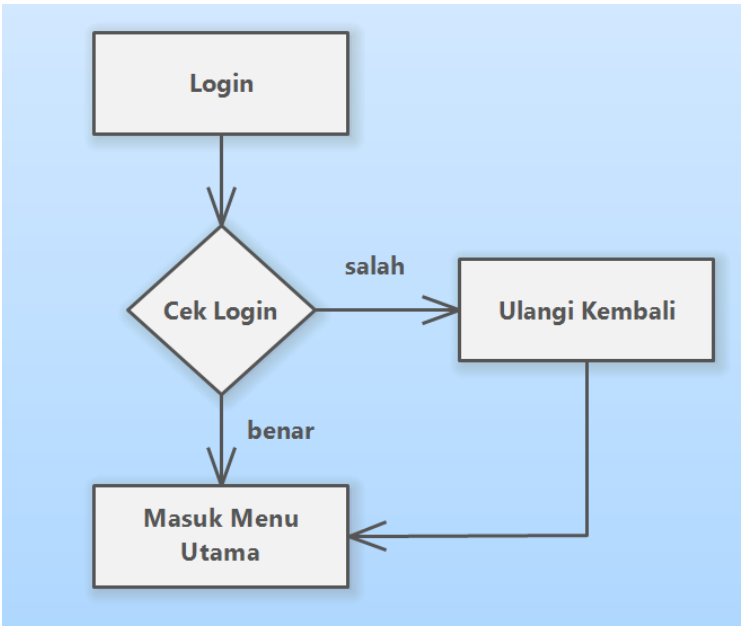
Ini disebut percabangan. Selain percabangan, struktur ini juga disebut *control flow*, *decision*, struktur kondisi, Struktur *if*, dan lain lain. Percabangan akan mampu membuat program berpikir dan menentukan tindakan sesuai dengan logika/kondisi yang kita berikan.

Pada buku ini, kita akan belajar struktur percabangan pada *Python*. Mulai dari yang paling dasar hingga yang kompleks.

Pastikan sebelumnya sudah paham tentang operator relasi dan logika.

A. Struktur Percabangan *If*

Percabangan *If* digunakan saat terdapat satu pilihan keputusan. Misalkan, kalau kita tidak lulus dalam ujian, maka kita ikut remedial. Sedangkan kalau lulus tidak perlu ikut remedial.



Gambar 7. 2 Tampilan Percabangan if

Maka kita bisa membuat kode-nya seperti ini:

```
if login == "benar":
    print("Selamat Datang di menu utama")
```

Kita menggunakan operator relasi sama dengan (==) untuk membandingkan isi variabel benar. Sedangkan tanda titik-dua (:) adalah tanda untuk memulai blok kode *If*.

Penulisan blok *If*, harus diberikan indentasi tab atau spasi 2x. Contoh penulisan yang salah:

```
if login == "benar":
print("Selamat Datang di menu utama ")
```

Contoh penulisan yang benar:

```
if login == "benar":
    print("Selamat Datang di menu utama ")
```

Contoh Program:

Biar pemahamannya semakin mantap, silahkan coba contoh kasus berikut ini.

```
# program untuk mengecek bonus dan diskon #
file: bonus.py
total_belanja = int(input("Total belanja: Rp "))
```

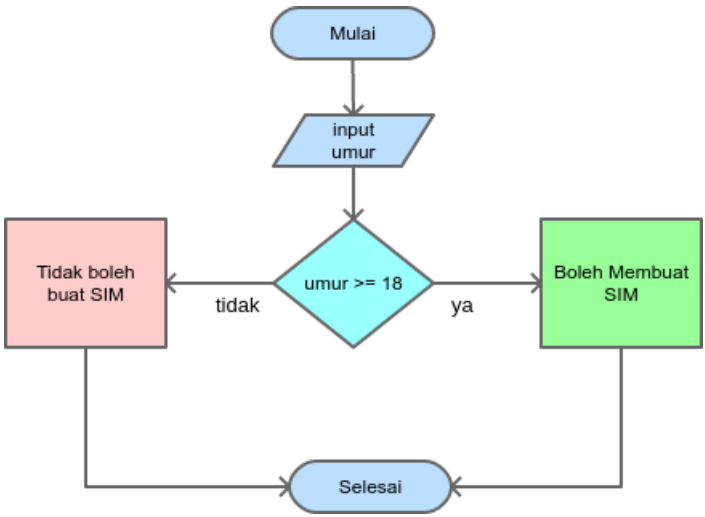
```
# tapi kalau dapat diskon akan berkurang bayar =
total_belanja
# jika dia belanja di atas 100rb maka berikan bonus dan diskon
if total_belanja > 100000:
    print("Kamu mendapatkan bonus minuman dingin") print("dan
    diskon 5%")
    # hitung diskonnya
    diskon = total_belanja * 5/100
    #5%
    bayar = total_belanja - diskon
    # cetak struk
    print("Total yang harus dibayar: Rp %s" % bayar)
    print("Terima kasih sudah berbelanja")
    print("Datang lagi yaa...")
```

Hasilnya:

Gambar 7. 3 Tampilan Hasil Percabangan if

B. Struktur Percabangan *If/Else*

Percabangan *If/Else* digunakan saat terdapat dua pilihan keputusan. Misalkan, jika umur diatas atau sama dengan 18 tahun boleh membuat SIM. Sedangkan dibawah itu belum boleh.



Gambar 7. 4 Tampilan Percabangan if/else

Maka kita bisa membuatnya dalam program:

```
# cek_umur.py
umur = int(input("Berapa umur kamu: "))

if umur >= 18:
  print("Kamu boleh membuat SIM")
else:
  print("Kamu belum boleh membuat SIM")
```

Hasil eksekusi dari kode di atas adalah sebagai berikut:

```
PROBLEMS 1  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\Arif Nurrochman\OneDrive\Documents\latihanpython> & C:/python/python.exe "c:/Users/Arif Nurrochman/OneDrive/Docum
nts/latihanpython/Main.py"
Berapa umur kamu: 13
Kamu belum boleh membuat SIM
PS C:\Users\Arif Nurrochman\OneDrive\Documents\latihanpython>
```

Gambar 7. 5 Tampilan hasil Percabangan if/else

C. Struktur Percabangan *If/Elif/Else*

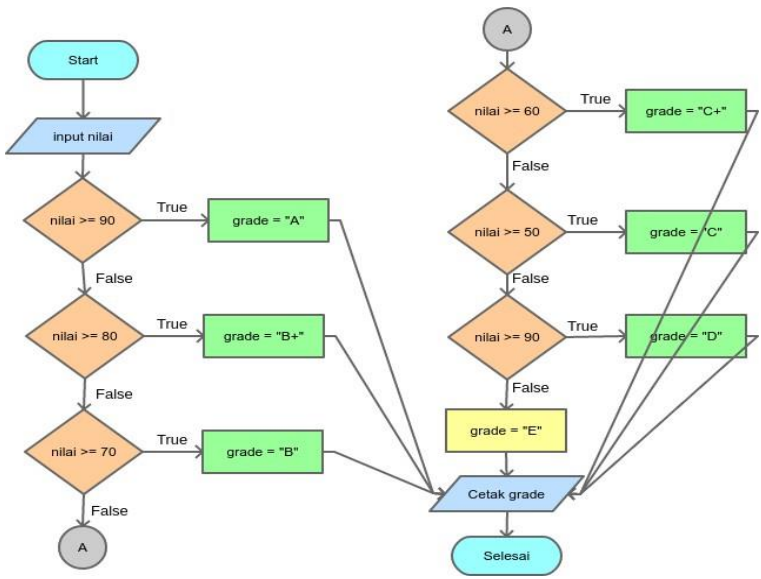
Percabangan *If/Elif/Else* digunakan apabila terdapat lebih dari dua pilihan keputusan.

```
if begini:
    maka ini
elif begitu:
    maka itu
else:
```

Kata kunci elif artinya *Else if*, fungsinya untuk membuat kondisi/logika tambahan apabila kondisi pertama salah.

Contoh Program:

Misalkan kita akan membuat program untuk menentukan grade nilai dengan *flow chart* sebagai berikut:



Gambar 7. 6 Tampilan Percabangan if/elseif/else

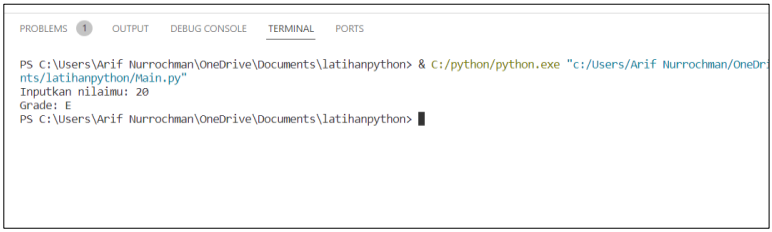
Maka kode programnya bisa kita buat seperti ini:

```
#file grade_nilai.py
nilai = int(input("Inputkan nilaimu: "))

if nilai >=
    90:
    grade =
    "A"
elif nilai >= 80:
    grade = "B+"
elif nilai >= 70:
    grade = "B"
elif nilai >= 60:
    grade = "C+"
elif nilai >= 50:
    grade = "C"
elif nilai >= 40:
    grade = "D"
else:
    grade = "E"

print("Grade: %s" % grade)
```

Maka hasilnya:



Gambar 7. 7 Tampilan hasil Percabangan if/elseif/else

D. Memahami Perulangan

Perulangan dalam bahasa pemrograman berfungsi menyuruh komputer melakukan sesuatu secara berulang-ulang. Terdapat dua jenis perualangan dalam bahasa pemrograman *Python*, yaitu perulangan dengan `for` dan `while`.

Perulangan `for` disebut *counted loop* (perulangan yang terhitung), sementara perulangan `while` disebut *uncounted loop* (perulangan yang tak terhitung). Perbedaannya adalah perulangan

`for` biasanya digunakan untuk mengulangi kode yang sudah diketahui banyak perulangannya. Sementara `while` untuk perulangan yang memiliki syarat dan tidak tentu berapa banyak perulangannya.

1. Perulangan for

Bentuk umum:

```
for indek in range(banyak_perulangan): #
        jalankan kode ini
        # jalankan juga kode ini
#kode ini tidak akan diulang karena berada di luar for
```

Contoh program:

```
# file: perulanganFor.py ulang = 10
for i in range(ulang):
        print ("Perulangan ke-"+str(i))
```

Pertama kita menentukan banyak perulangannya sebanyak 10x

```
ulang = 10
```

Variabel `i` berfungsi untuk menampung indeks, dan fungsi `range()` berfungsi untuk membuat list dengan range dari 0-10. Fungsi `str()` berfungsi merubah tipe data ineger ke string.

```
for i in range(ulang):
        print "Perulangan ke-"+str(i)
```

Hasil:

Selamatkode@imajinasi:~$ *Python* perulanganFor.py
Perulangan ke-0
Perulangan ke-1
Perulangan ke-2
Perulangan ke-3

Perulangan ke-4
Perulangan ke-5
Perulangan ke-6
Perulangan ke-7
Perulangan ke-8
Perulangan ke-9

Contoh lain menggunakan senarai (*list*):

 # berkas: perulanganFor.py

```
1   item = ['jeruk','semangka','apel','anggur']
2   for isi in item:
3       print (isi)
4
```

Hasil:

```
PS C:\Users\Arif Nurroc
ocuments/latihanpython_
jeruk
semangka
apel
anggur
```

2. Perulangan while

Bentuk umum:

```
while(True):

        # jalankan kode ini

# kode ini berada di luar perulangan while
```

Contoh:

```
1   jawab = 'ya'
2   hitung = 0
3   while(jawab == 'ya'):
4       hitung += 1
5       jawab = input("Ulang lagi tidak? ")
6       print ("Total perulagan: " + str(hitung))
7
```

Atau bisa juga dengan bentuk yang seperti ini, dengan menggunakan kata kunci `break`

```
# berkas: perulanganWhile.py
jawab = 'ya'
hitung = 0
while(True):
    hitung += 1
    jawab = input("Ulang lagi tidak? ")
    if jawab == 'tidak':
        break
    print ("Total perulagan: " + str(hitung))

```

Pertama menentukan variabel untuk menghitung, dan menentukan kapan perulangan berhenti. kalau pengguna menjawab tidak maka perulangan akan terhenti.

jawab = 'ya'

hitung = 0

Melakukan perulangan dengan *while,* kemudian menambah satu variabel `hitung` setiap kali mengulang. lalu menanyakan kepada pengguna, apakah mau berhenti mengulang atau tidak?

```
# berkas: perulanganWhile.py
jawab = 'ya'
hitung = 0
while(jawab == 'ya'):
    hitung  += 1
    jawab = input("Ulang lagi tidak? ")

```

Setelah selesai mengulang, cetak berapa kali perulangan tersebut terjadi

print "Total perulagan: " + str(hitung)

Hasil:

```
PS C:\Users\Arif Nurroch
ocuments/latihanpython_
Ulang lagi tidak? ya
Ulang lagi tidak? ya
Ulang lagi tidak? ya
Ulang lagi tidak? ya
```

BAB 8
Fungsi dan Prosedur pada *Python*

Dalam pembuatan program yang kompleks dan memiliki banyak fitur, kita diharuskan menggunakan fungsi dan prosedur. Kenapa memangnya kalau tidak menggunakan fungsi? Bisa jadi kita akan mengalami kesulitan menulis kode programnya, karena banyak yang harus ditulis dan kode akan menjadi sulit dibaca dan dirawat *(maintenance)*.

Dengan fungsi, kita dapat memecah program besar menjadi sub program yang lebih sederhana. Masing-masing fitur pada program dapat kita buat dalam satu fungsi. Pada saat kita membutuhkan fitur tersebut, kita tinggal panggil fungsinya saja.

Hal ini akan kita coba pada contoh program yang sudah tersedia dibawah ini. Sebelum lihat program alangkah baiknya kita memahami teori dasar dan hal apa saja yang harus kita ketahui tentang fungsi di *Python*.

A. Cara Membuat Fungsi pada Python

Fungsi pada *Python*, dibuat dengan kata kunci `def` kemudian diikuti dengan nama fungsinya.

Contoh:

```
def nama_fungsi():
    print "Hello ini Fungsi"
```

Sama seperti blok kode yang lain, kita juga harus memberikan identasi (tab atau spasi 2x) untuk menuliskan isi fungsi.

Setelah kita buat fungsinya, kita bisa memanggilnya seperti ini:

```
nama_fungsi()
```

Sebagai contoh, coba tulis kode program berikut:

```
1  # Membuat Fungsi
2  def salam():
3      print ("Selamat Datang Mahasiswa Global Institute")
4  ## Pemanggilan Fungsi
5  salam()
6
```

Hasilnya:

```
Selamat Datang Mahasiswa Global Institute
```

Bagaimana? mudah bukan, intinya apapun yang ada di dalam fungsi, ketika dipanggil itulah yang akan dilakukan.

FYI: fungsi juga dapat dipanggil pada fungsi lain, bahkan bisa memanggil dirinya sendiri. Fungsi yang memanggil dirinya sendiri, disebut *fungsi rekursif*.

B. Fungsi dengan Parameter

Buku ini membahas bagaimana kalau kita ingin memberikan input nilai ke dalam fungsi. Dengan memanfaatkan parameter. Parameter adalah variabel yang menampung nilai untuk diproses di dalam fungsi.

Contoh:

```
def salam(ucapan):
    print(ucapan)
```

Pada contoh diatas, kita akan membuat fungsi dengan parameter ucapan.

Cara pemanggilan fungsi yang memiliki parameter adalah seperti ini:

salam("Selamat Datang Mahasiswa Global Institute")

"Selamat siang" → adalah nilai parameter yang akan diberikan.

Lalu bagaimana kalau parameternya lebih dari satu.

Kita bisa menggunakan tanda koma (,) untuk memisahnya. Contoh:

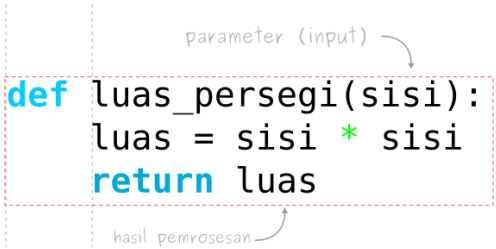
```
1   # Membuat fungsi dengan parameter
2   def luas_persegipanjang(panjang, lebar):
3       luas = panjang * lebar
4       print ("Luas Persegi Panjang: %f")%luas;
5   # Pemanggilan fungsi
6       luas_persegipanjang(4, 6)
```

Hasilnya:
Luas persegi panjang: 24.000000

C. Fungsi yang Mengembalikan Nilai

Fungsi yang tidak mengembalikan nilai biasanya disebut dengan prosedur. Namun, kadang kita butuh hasil proses dari fungsi untuk digunakan pada proses berikutnya.

Maka fungsi harus mengembalikan nilai dari hasil pemrosesannya. Cara mengembalikan nilai adalah menggunakan kata kunci return lalu diikuti dengan nilai atau variabel yang akan dikembalikan. Berikut struktur parameternya.



Contoh:

```
def luas_persegi(sisi): luas =
    sisi * sisi return luas
# pemanggilan fungsi
print ("Luas persegi: %d" % luas_persegi(6))
```

Hasilnya:
Luas persegi: 36

Perbedaan dengan fungsi `luas_persegipanjang()` yang tadi. Pada fungsi `luas_ persegipanjang()` kita melakukan print dari hasil pemrosesan secara langsung di dalam fungsinya.

Sedangkan fungsi `luas_persegi()`, kita melakukan print pada saat pemanggilannya. Jadi, fungsi `luas_persegi()` akan bernilai sesuai dengan hasil yang dikembalikan.

Sehingga kita dapat memanfaatkannya untuk pemerosesan berikutnya.

Misalnya seperti ini:

```
# rumus: sisi x sisi def
luas_persegi(sisi):
    luas = sisi * sisi return
    luas
# rumus: sisi x sisi x sisi def
volume_persegi(sisi):
    volume = luas_persegi(sisi) * sisi
```

Pada contoh di atas, kita melakukan pemanggilan fungsi `luas_persegi()` untuk menghitung volume persegi.

D. Variabel Global dan Lokal pada *Python*

Saat kita menggunakan fungsi, maka kita juga harus mengetahui yang namanya variabel Global dan Lokal. Yang maksud variabel Global adalah variabel yang bisa diakses dari semua fungsi, sedangkan variabel lokal hanya bisa diakses di dalam fungsi tempat dia berada saja. Pada *Python*, urutan pengaksesan variabel *(scope)* dikenal dengan sebutan LGB (*Local, Global, dan Build-in*).

Jadi program *Python* mulai mencari vairabel lokal terlebih dahulu, kalau ada maka itu yang digunakan. Tapi kalau tidak ada, pencarian terus ke Global, dan Build-in.

Variabel Build-in adalah variabel yang sudah ada di dalam *Python*.

Contoh program:

```
1   def help():
2   # ini variabel lokal
3       nama = "Global Institute"
4       versi = "1.0.2"
5   # mengakses variabel lokal
6       print ("Nama: %s" % nama)
7       print ("Versi: %s" % versi)
8   # mengakses variabel global
9       print ("Nama: %s" % nama)
10      print ("Versi: %s" % versi)
11  # memanggil fungsi help()
12      help()
```

Hasilnya:
Nama: Global Institute
Versi: 1.0.2

Perhatikanlah variabel nama yang berada di dalam fungsi `help()` dan diluar fungsi `help()`. Variabel nama yang berada di dalam fungsi `help()` adalah variabel lokal.

Jadi, saat kita memanggil fungsi `help()` maka nilai yang akan tampil adalah nilai yang ada di dalam fungsi `help()`.

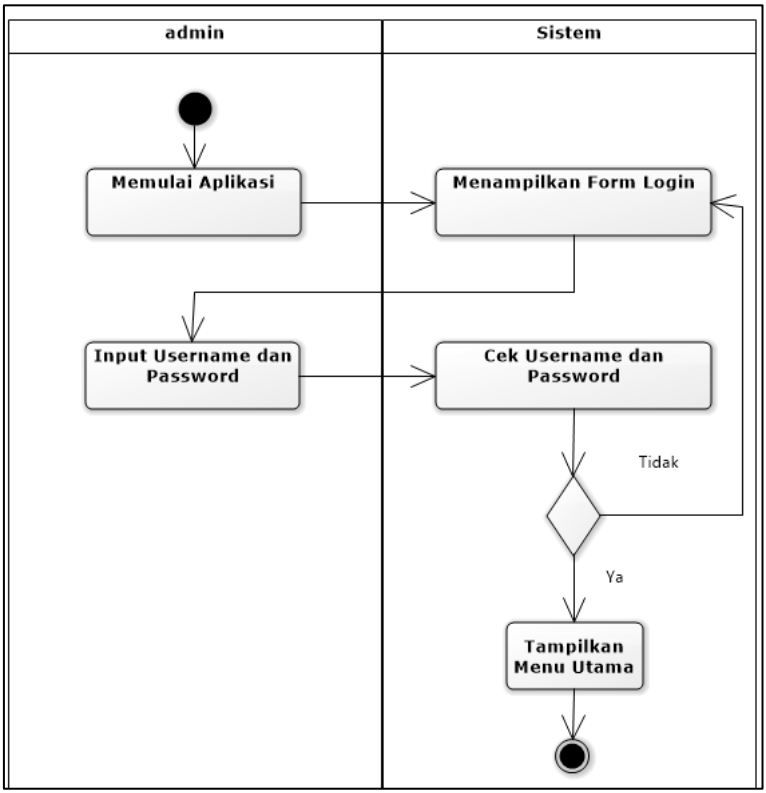
Kenapa tidak tampil yang global karena *Python* mulai mencari dari lokal, ke global, dan build-in. Kalau di tiga tempat itu tidak ditemukan, maka biasanya akan terjadi *NameError* atau variabel tidak ditemukan.

E. Contoh Program dengan Fungsi

Sekarang kita memulai program, pertama tama silahkan buat file baru bernama program_fungsi.py.

Pertama kita buat sebuah variabel global berupa list untuk menampung judul-judul buku.

Berikut flow chart sistem login :



Berikut codingnya :

```
# membuat fungsi authentifikasi sederhana
def get_login():
    print('=' * 20)
    print('halaman login penggajian')
    username = input('masukan username kasir anda: ')
    password = input('masukan password: ')

    if username == 'admin' and password == 'adminpass':
        print('login berhasil...\n\n')
        main_menu()
    else:
        print('login gagal coba lagi..')
        get_login()
```

```
def main_menu():
    # membuat daftar menu pada kasir
    print('=' * 10, 'MAIN MENU APLIKASI PENGGAJIAN', '=' * 10)
    print('selamat datang di aplikasi penggajian')
```

```
# main program
if __name__=='__main__':
    get_login()
```

Hasilnya adalah :

```
halaman login penggajian
masukan username kasir anda: admin
masukan password: adminpass
login berhasil...


========== MAIN MENU APLIKASI PENGGAJIAN ==========
selamat datang di aplikasi penggajian
```

BAB 9
Variabel dan Tipe Data dalam *Python*

Pada kesempatan ini kita akan membahas tentang variabel dan tipe data pada *Python*. Pastikan sebelumnya sudah mengetahui cara membuat skrip atau program *Python*.

A. Pengertian Variabel dan Tipe Data

Variabel adalah lokasi memori yang dicadangkan untuk menyimpan nilai-nilai. Ini berarti bahwa ketika Anda membuat sebuah variabel Anda memesan beberapa ruang di memori.
Variabel bersifat *mutable*, artinya nilainya bisa berubah-ubah.

B. Membuat Variabel di *Python*

Variabel di *Python* dapat dibuat dengan format seperti ini: nama_variabel = <nilai>
Contoh:

```
variabel = "isi variabel"
variabel2 = 20
```

Kemudian untuk melihat isi variabel, kita dapat menggunakan fungsi `print`.
print variabel_ku dan print variabel2
Aturan Penulisan Variabel

1. Nama variabel boleh diawali menggunakan huruf atau garis bawah (_), contoh: nama, _nama, namaKu, nama_variabel.
2. Karakter selanjutnya dapat berupa huruf, garis bawah (_) atau angka, contoh: __nama, n2, nilai1.
3. Karakter pada nama variabel bersifat sensitif (*case- sensitif*). Artinya huruf besar dan kecil dibedakan. Misalnya, variabel_Ku dan variabel_ku, keduanya adalah variabel yang berbeda.
4. Nama variabel tidak boleh menggunakan kata kunci yang sudah ada dalam *Python* seperti if, while, for, dsb.

C. Menghapus Variabel

Ketika sebuah variabel tidak dibutuhkan lagi, maka kita bisa menghapusnya dengan fungsi `del()`.

Contoh:

```
>>> nama = "global institute"
>>> print nama
globalinstitute
>>> del(nama)
>>> print nama
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'nama' is not defined
```

Pada perintah terakhir, kita akan mandapatkan `NameError`. Artinya variabel tidak ada di dalam memori alias sudah dihapus.

1. Tipe data

Cara mengisi nilai variabel ditentukan dengan jenis datanya, misalkan untuk tipe data teks *(string)* maka harus diapit dengan tanda petik ("..."). Sedangkan untuk angka *(integer)* dan *boolean* tidak perlu diapit dengan tanda petik.

Contoh:

nama_ku = "Glow"
usia = 18
berat = 83.22

Python akan seara otomatis mengenali jenis data atau tipe data yang tersimpan dalam sebuah variabel. Untuk memeriksa tipe data pada suatu variabel, kita bisa menggunakan fungsi `type()`.

Contoh:

```
>>> usia = 18
>>> type(usia)
<type 'int'>
>>> usia = "18"
>>> type(usia)
<type 'str'>
```

```
>>> usia = '18'
>>> type(usia)
<type 'str'>
>>> usia = 18.5
>>> type(usia)
<type 'float'>
>>> usia = true
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
NameError: name 'true' is not defined
>>> usia = True
>>> type(usia)
<type 'bool'>
```

2. Jenis-jenis Tipe Data

Secara umum, tipe data primitif dalam *Python* dibagi menjadi tiga jenis:

a. Tipe data angka
b. Tipe data teks
c. Tipe data boolean

Mari kita bahas satu per satu…

a. Tipe Data Angka
   Tipe data angka dibagi menjadi beberapa jenis lagi:
   1) int (Integer): bilangan bulat, contoh 32, 22, 12, 10, dsb.
   2) float: bilangan pecahan, contoh 1.3, 4.2, 22.3, dsb.

Contoh:

```
luas = 120 #tipe int
berat = 50.12 #float
jarak = 1e2 #float 3000.0,
huruf e artinya eksponen 10
```

b. Tipe Data Teks

Tipe data teks dibagi menjadi dua jenis lagi:

1) Char: Karakter, contoh 'R'.
2) String: Kumpulan karakter, contoh "aku lagi makan". Penulisan tipe data teks harus diapit dengan tanda petik. Bisa menggunakan petik tunggal ('...'), ganda ("..."), dan tiga ('''...'''atau """..."""").

Contoh:

```
nama = "Ivan"
jenis_kelamin = 'L'
alamat = """Jl. Melati I Rt 06 Rw 10 Kode Pos 17837,
Kelurahan Waluya , Kab Bekasi""
agama = 'islam'
```

c. **Tipe data boolean**

Tipe data *boolean* adalah tipe data yang hanya memiliki dua nilai yaitu Truedan Falseatau 0dan 1.
Penulisan Truedan False, huruf pertamnya harus kapital dan tanpa tanda petik.

Contoh:

```
bergerak = True
nyala = 1 #sebenarnya tipenya int, tapi bisa juga menjadi bool
```

D. Contoh Program Variabel dan Tipe Data

Berikut ini contoh sederhana penerapan variabel dalam program.

```
1   # Mengambil input
2   nama = input("Siapa nama kamu: ")
3   umur = input("Berapa umur kamu: ")
4   # Menampilkan output
5   print ("Hello",nama,"umur kamu adalah",umur,"tahun")
```

Hasilnya :

```
PS C:\Users\Arif Nurrochman\OneDrive\Documents\latihanpython_cA23> & C:/
ocuments/latihanpython_cA23/Main.py"
Siapa nama kamu: Global Institute
Berapa umur kamu: 8
Hello Global Institute umur kamu adalah 8 tahun
PS C:\Users\Arif Nurrochman\OneDrive\Documents\latihanpython_cA23>
```

E. Konversi Tipe Data

Meskipun *Python* telah otomatis mendeteksi tipe data yang tersimpan dalam variabel, tapi ada kalanya kita perlu melakukan konversi tipe data.

Misalkan, pada contoh berikut ini:

```
a = 9
b = 3
c = a / b
print (c)
#output: 3
```

Pembagian nilai a dan b menghasilkan 3 (integer). Mengapa demikian?

Karena nilai a dan b bertipe integer, maka hasilnya pun berupa integer.

Bagaimana agar hasilnya ada komanya?

Tentu kita harus merubah tipe variabel a dan b menjadi bilangan pecahan (float) dulu, baru setelah itu dibagi.

```
a = 10
b = 3
c = float(a) / float(b)
print (c)
#output: 3.3333333333333335
```

Fungsi `float()` akan mengubah nilai a menjadi 10.0 dan b menjadi `3.0`.

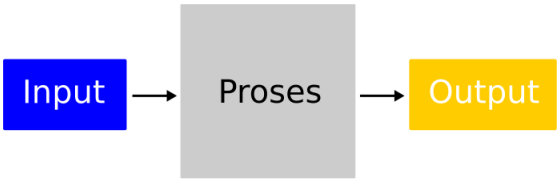
Fungsi-fungsi untuk mengubah tipe data:

1. *int*() untuk mengubah menjadi integer;
2. *long*() untuk mengubah menjadi integer panjang;
3. *float*() untuk mengubah menjadi float;
4. *bool*() untuk mengubah menjadi boolean;
5. *chr*() untuk mengubah menjadi karakter;
6. *str*() untuk mengubah menjadi string.
7. *bin*() untuk mengubah menjadi bilangan Biner.
8. *hex*() untuk mengubah menjadi bilangan Heksadesimal.
9. *oct*() untuk mengubah menjadi bilangan okta.

BAB 10
Mengambil Input dan Menampilkan Output

Input adalah masukan yang kita berikan ke program. Program akan memprosesnya dan menampilkan hasil outputnya.

Input, proses, dan output adalah inti dari semua program komputer.



Gambar 10. 1 Tampilan Flow Input output

Pada buku ini, kita akan belajar cara mengambil input dan menampilkan output untuk program berbasis teks.

A. Cara Mengambil Input dari Keyboard

Python sudah menyediakan fungsi `input()` dan `input()` untuk mengambil inputan dari keyboard. nama_varabel = input("Sebuah Teks")

Artinya, teks yang kita inputkan dari keyboard akan disimpan ke dalam nama_variabel.

```
1   kode_buku = str(input('Kode Buku : '))
2   judul = str(input('Judul Buku : '))
3   pengarang = str(input('Pengarang Buku : '))
4   penerbit = str(input('Penerbit Buku : '))
5   kategori = str(input('Kategori Buku : '))
6   tahun_terbit = int(input('Tahun Terbit : '))
7
```

```
8   #menampilkan output
9   print(kode_buku)
10  print(judul)
11  print(pengarang)
12  print(penerbit)
13  print(kategori)
14  print(tahun_terbit)
```

Hasilnya:

```
Kode Buku : A109
Judul Buku : Belajar Python
Pengarang Buku : Global Institute
Penerbit Buku : Global Institute Library
Kategori Buku : Pendidikan
Tahun Terbit : 2023
A109
Belajar Python
Global Institute
Global Institute Library
Pendidikan
2023
PS C:\Users\Arif Nurrochman\OneDrive\Documents\latihanpython cA23>
```

Apa perbedaan fungsi `str(input)` dengan `int(input)`?
Fungsi `int(input)` digunakan untuk mengambil data angka. Sedangkan `str(input)` untuk mengambil teks.

B. Cara Menampilkan Output

Seperti yang kita sudah ketahui pada contoh-contoh sebelumnya. Untuk menampilkan output teks, kita
Menampilkan Variabel dan Teks
Pada contoh di atas kita menggunakan tanda koma (,) untuk menggabungkan teks dan variabel yang akan ditampilkan.

```
nama = ("Selamat Datang")
print ("Hello",nama)
```

Hasil:
Hello Selamat Datang
Antara kata `Hello` dan `Globalinstitute` terdapat spasi sebagai pemisah, karena kita menggunakan tanda koma.

Jangan ditambahkan kurung seperti ini:

```
nama = "Global Institute"
print("Hello",nama)
```

Karena akan dibaca sebagai Tuple yang akan mengahsilkan output seperti ini:
('Hello', 'Global Institute')
Sebaiknya jangan dikurung kalau menggunakan tanda koma. Jika ingin menggunakan kurung, maka kita harus menggabungkan teks dan variabelnya dengan tanda plus (+).

Contoh:

```
nama = " Global Institute
" print("Hello " + nama)
```

Hasilnya:
Hello Global Institute

Menggunakan Fungsi format()
Fungsi `format()` akan menggabungkan isi variabel dengan teks.
Contoh:

```
nama = input("Nama: ")
print ("Selamat {} menjadi juara 1".format(nama))
```

Tanda {} akan otomatis diganti sesuai dengan nilai yang kita inputkan ke variabel `nama` .
Contoh lagi:
String Formatting Cara Lama

```
22  # Penggabungan
23  nama_mu = input("Nama kamu: ")
24  nama_dia = input("Nama dia: ")
25  print ("{} dengan {} sepertinya pasangan yang serasi:)".format(nama_mu, nama_dia))
```

Hasil :

```
Nama kamu: arif
Nama dia: nisa
arif dengan nisa sepertinya pasangan yang serasi:)
PS C:\Users\Arif Nurrochman\OneDrive\Documents\latihanpython cA23>
```

Menggunakan teks dan variabel cara lama menggunakan simbol persen (%).

Contoh:

```
nama = input("Inputkan nama buah: ")
print ("Buah ini bernama buah %s" % nama)
```

Tanda %sakan otomatis diganti dengan nilai yang kita inputkan ke variabel nama.

BAB 11
Jenis Operator dalam *Python*

Setelah kita mengenal variabel dan tipe data pada *Python*, selanjutnya kita akan berkenalan dengan Operator. Operator merupakan simbol-simbol yang digunakan untuk melakukan operasi tertentu.

Ada enam jenis operator dalam pemrograman bahasa *Python* yang harus diketahui:

1. Operator Aritmatika
2. Operator Pembanding/Relasi
3. Operator Penugasan
4. Opeartor Logika
5. Operator Bitwise
6. Operator Ternary

A. Operator Aritmatika

Operator aritmatika pada *Python* adalah operator yang digunakan untuk melakukan perhitungan data. Berikut pengertian dan contoh penggunaan operator aritmatika pada *Python*. Catatan: ">>>" merupakan perunjuk interpreter *Python*.

Operator	Simbol	Penjelasan	Contoh Interpreter
Penjumlahan	+	Menjumlahkan masing- masing bilangan (operan)	>>> 1+2 3
Pengurangan	–	Mengurangi nilai operan kiri dengan operan kanan	>>> 4-7 -3
Perkalian	*	Mengalikan masing-masing bilangan (operan)	>>> 7*4 28
Pembagian	/	Membagi nilai operan kiri dengan operan kanan	>>> 9/5 1.8
Pembagian bulat	//	Menghilangkan nilai desimal suatu pembagian	>>> 9//5 1

Operator	Simbol	Penjelasan	Contoh Interpreter
Modulo	%	Menghasilkan sisa pembagian	>>> 14%5 4
Eksponen	**	Perpangkatan dengan operan kiri sebagai pokok	>>> 4**3 64

Berikut contoh penggunaan operator aritmatika pada *Python* shell

Mari kita coba dalam program *Python*nya

```
# file: operator_aritmatika.py
# Ambil input untuk mengisi nilai
x = input("Inputkan nilai x: ")
y = input("Inputkan nilai y: ")

# Menggunakan operator penjumlahan
z = x + y
print ("Hasil %d + %d = %d" % (x,y,z))


# Operator
Pengurangan z = x - y
print ("Hasil %d - %d = %d" % (x,y,z))


# Operator
Perkalian z = x *
y
print ("Hasil %d * %d = %d" % (x,y,z))
```

```
# Operator Pembagian z = x / y
print ("Hasil %d / %d = %d" % (x,y,z))


# Operator Sisa Bagi z = x % y
print ("Hasil %d %% %d = %d" % (x,y,z))


# Operator Pangkat z = x ** y
print ("Hasil %d ** %d = %d" % (x,y,z))
```

Pada kode program diatas, kita menggunakan *string formatting* untuk mencetak hasil dari masing-masing opearasi. Operator % selain digunakan untuk *string formatting*, operator ini

juga digunakan untuk menghitung operasi sisa bagi.
Misal: 5 % 2, maka hasilnya 1. Karena sisa dari hasil bagi antara 5 dengan 2 adalah 1.

B. Operator Penugasan

Seperti namanya, operator ini digunakan untuk memberikan tugas pada variabel.
Misalnya:
umur = 18

Maka variabel umur telah kita berikan tugas untuk menyimpan angka 18. Selain menyimpan atau pengisian nilai, ada juga menjumlahkan, mengurangi, perkalian, pembagian, Dan sebagainya.

Operator penugasan adalah operator yang digunakan untuk memasukkan nilai atau memodifikasi nilai suatu variabel.

Operator	Simbol	Penjelasan	Contoh Interpreter
Sama dengan	=	Mendefinisikan nilai suatu varibel.	>>> x=3

Tambah sama dengan	+=	Menambahkan nilai suatu variabel	>>> y=2
			>>> y+=7
			>>> y
			9

Kurang sama dengan	-=	Mengurangkan nilai suatu variabel	>>> z=4
			>>> z-=2
			>>> z
			2

Kali sama dengan	*=	Mengalikan nilai suatu variabel	>>> a=6
			>>> a*=3
			>>> a
			18

Bagi sama dengan	/=	Membagi nilai suatu variabel	>>> b=8
			>>> b/=4
			>>> b
			2.0

Pembagian bulat sama dengan	//=	Pembagian bulat nilai suatu variabel	>>> c=5
			>>> c//=2
			>>> c
			2

Operator	Simbol	Penjelasan	Contoh Interpreter

Modulo sama dengan	%=	Sisa pembagian nilai variabel dengan suatu bilangan	>>> d=7
			>>> d%=3
			>>> d
			1

Pangkat sama dengan	**=	Perpangkatan nilai suatu variabel	>>> e=2
			>>> e**=3
			>>> e
			8

C. Operator Pembandingan

Operator perbandingan pada *Python* digunakan untuk melakukan suatu perbandingan antar operan. Output yang ditampilkan adalah nilai boolean *True* atau *False*. Operator perbandingan sering digunakan dalam perulangan. Berikut pengertian dan contoh penggunaan operator perbandingan pada *Python*.

Operator	Simbol	Penjelasan	Contoh Interpreter
Persamaan	==	Bernilai True: operan mempunyai nilai yang sama Bernilai False: operan tidak mempunyai nilai yang sama	>>> 1==1.0 True>>> 1==4 False
Tidak sama	!=	Bernilai True: operan mempunyai nilai yang tidak sama Bernilai False: operan mempunyai nilai yang sama	>>> 7!=8 True>>> 2!=2.0 False
Lebih besar dari	>	Bernilai True: operan kiri lebih besar dari operan kanan Bernilai False: operan kiri tidak lebih besar dari operan kanan	>>> 9>7 True>>> 9>9 False
Lebih kecil dari	<	Bernilai True: operan kiri lebih kecil dari operan kanan Bernilai False: operan kiri tidak lebih kecil dari operan kanan	>>> 7<8 True>>> 8<7 False

Operator	Simbol	Penjelasan	Contoh Interpreter
Lebih besar atau sama dengan	>=	Bernilai True: operan kiri lebih besar atau sama dengan operan kanan Bernilai False: operan kiri tidak lebih besar atau sama dengan operan kanan	>>> 9>=7 True>>> 7>=9 False
Lebih kacil atau sama dengan	<=	Bernilai True: operan kiri lebih kecil atau sama dengan operan kanan Bernilai False: operan kiri tidak kecil besar atau sama dengan operan kanan	>>> 8<=8 True>>> 7<=6 False

Contoh :

a = 8

b = 9

c = a < b

Apakah isi dari variabel c

Isinya adalah True, karena nilai 8 lebih kecil dari 9 (8 < 9) adalah salah (True).

Untuk lebih jelasnya, kita coba contonya dalam program.

```
# file: operator_pembanding.py
a = input("Inputkan nilai a: ")
b = input("Inputkan nilai b: ")

print ("a > b \t:",a > b) #lebihbesar
print ("a < b \t:",a < b) #lebihkecil
print ("a == b \t:",a == b) #samadengan
print ("a != b \t:",a!=b) #tidaksamadengan
print ("a >= b \t:",a>= b) #lebihbesarsamadengan
print ("a <= b \t:",a<= b) #lebihkecilsamadengan
```

D. Operator Logika

Operator logika digunakan untuk membuat operasi logika, seperti logika AND, OR, dan NOT.

Operator logika terdiri dari:

Nama	**Simbol di *Python***
Logika AND	And
Logika OR	Or
Negasi/kebalikan	not

Logika OR

X	Y	HASIL
TRUE	TRUE	TRUE
TRUE	FALSE	TRUE
FALSE	TRUE	TRUE
FALSE	FALSE	FALSE

Logika AND

X	Y	HASIL
TRUE	TRUE	TRUE
TRUE	FALSE	FALSE
FALSE	TRUE	FALSE
FALSE	FALSE	FALSE

Contoh programnya sebagai berikut:

```
a = True
b = False

# Logika
AND c =
a and b
print "%r and %r = %r" % (a,b,c)

# Logika OR
c = a or b
print "%r or %r = %r" % (a,b,c)

# Logika
Not c =
not a
print "not %r  = %r" % (a,c)
```

E. Operator Bitwise

Operator Bitwise adalah operator untuk melakukan operasi berdasarkan bit/biner.

Operator ini terdiri dari:

Nama	Simbol
AND	&
OR	|
XOR	^
Negasi/kebalikan	~
Left Shift	<<
Right Shift	>>

Hasil operasi dari operator ini agak sulit dipahami, kalau kita belum paham operasi bilangan biner. Sekarang coba pahami dengan contoh sederhana:

Misalnya, kita punya variabel a = 60 dan b = 13.

Bila dibuat dalam bentuk biner, akan menjadi seperti ini:

a = 00111100

b = 00001101

Kemudian, dilakukan operasi bitwise

Operasi AND

a = 00111100

b = 00001101

a & b = 00001100

Operasi OR

a = 00111100

b = 00001101

a | b = 00111101

Operasi XOR

a = 00111100

b = 00001101

a ^ b = 00110001

Opearsi NOT (Negasi/kebalikan) a = 00111100
~a = 11000011

Konsepnya memang hampir sama dengan opeartor Logika. Namun, Bitwise digunakan untuk biner.
Mari kita coba dalam program…

```
x = 15
y = 12

print('x berisi angka',x ,'desimal')
print('y berisi angka',y ,'desimal')

print('\n')

print('x & y  :',x & y)
print('x | y  :',x | y)
print('x ^ y  :',x ^ y)
print('~x     :',~x)
print('x << 1 :',x << 1)
print('x >> 1 :',x >> 1)
```

F. Operator Ternary

Operator ternary juga dikenal dengan operator kondisi, karena digunakan untuk membuat sebuah ekspresi kondisi seperti percabangan IF/ELSE.

Operator ternary sebenarnya tidak ada dalam *Python*, tapi *Python* punya cara lain untuk menggantikan operator ini.

Pada bahasa pemrograman lain operator ternary menggunakan tanda tanya (?) dan titik dua (:). kondisi ? <nilai true> : <nilai false>. Berikut Contohnya.

```
aku = (umur < 10) ? "bocah" : "dewasa"
```

Dalam *Python* bentuknya berbeda, yaitu menggunakann IF/ELSE dalam satu baris.

```
<Nilai True> if Kondisi else <Nilai False>
```

Contoh:

```
umur = input("berapa umur kamu? ")
aku = "bocah" if umur < 10 else "dewasa" print aku
```

Silakan dicoba untuk mengisi nilai variabel umur dengan nilai di bawah 10 dan perhatikan output-nya. Cara lain untuk membuat operasi ternary juga bisa menggunakan *Tuple* dan *List*.

```
jomblo = True
status = ("Menikah", "Single")[jomblo]
print (“status”)
```

BAB 12
Penulisan dan Penggunaan String pada *Python*

String pada *Python* adalah tipe data yang memuat satu karakter atau lebih karakter (*sequences of character*) yang diapit oleh tanda petik tunggal („) atau tanda petik (“). Dalam bahasa pemrograman *Python*, deklarasi suatu string tidak dibedakan penggunaan tanda petik atau tanda petik tunggal. Hal ini berbeda dengan bahasa pemrograman yang mendekati bahasa mesin seperti C++ dan C, dimana penggunaan („) untuk mendefinisikan satu karakter dan (“) untuk mendefiniskan banyak karakter.

"Hello World"
'Hello World'

A. Penulisan String pada *Python*

Untuk menulis string serta menampilkannya langsung, dapat menggunakan syntax

print(“Hello World”)

Sistematika menampilkan string dengan syntax *prnit()*

1. Syntax *print()* digunakan untuk menampilkan string berupa output pada console.
2. Didalam syntax dituliskan string atau variabel.

Beberapa karakter khusus pada *Python*

Karakter	Fungsi
\n	Untuk membuat New Line atau paragraf baru
\s	Untuk membuat karakter spasi
\e	Untuk membuat karakter escape
\b	Untuk membuat backspace
\t	Untuk membuat tabulasi

Misalkan ditampilkan suatu string *Hello World*, dapat diilustrasikan sebagai berikut

B. Mendefinisikan Variabel String pada *Python*

Untuk mendefinisikan variabel string pada *Python*, praktiknya hampir sama dengan mendefinisikan variabel numerik. Misalkan dibuat suatu variabel x yang berisi nilai string "Hello World"

```
1 x = "Hello World"
1 print(x)
```

Catatan: Syntax **;** digunakan untuk membuat line baru pada console.

C. Operasi String pada *Python*

String pada *Python* juga dapat dilakukan eksekusi secara komputasi. Misalnya perulangan dan penjumlahan string.

Contoh: *Penjumlahan String (+)*

```
1  a = "Global "
2  b = " Institute"
3  c = a+b
4  print (c)
```

Berikut ilustrasi operasi penjumlahan string

```
1   x = "Belajar"
2   y = "Python"
3   c = x+y
4   print(c)
5
```

Contoh: *Perkalian String (*)*

```
1  x = "Python "
2  print (x*7)
```

```
1   y = "Python "
2   print(y*7)
3
```

```
Python Python Python Python Python Python Python
PS C:\Users\Arif Nurrochman\OneDrive\Documents\latihanpython_cA23>
```

D. Mengindex String pada *Python*

Variabel yang menyimpan suatu nilai berupa barisan karakter, sehingga dapat dilakukan index string untuk menampilkan karakter dengan index tertentu. Index String menggunakan bilangan bulat dari 0.

nama_variabel[index_karakter]

Misalnya akan ditampilkan karakter dengan index 1 pada suatu variabel string **x** berikut.

1 x = "Hello World"
2 print (x[2])

```
1   x = "Hello World"
2
3   print (x[2])
```

terlihat variabel x memuat string "Hello World", dimana x[0] = "H", x[1] = "e", x[2] = "l", dan seterusnya.

E. Range Slice pada Variabel String *Python*

Range Slice menampilkan range karakter dari a mendekati b (limit b), yang diformulasikan dengan,

nama_variabel[a:b]

a = index karakter yang mulai dicetak

b = batas akhir karakter yang dicetak, b tidak dicetak atau kosongkan untuk mencetak sampai karakter terakhir.

Misalnya pada variabel x akan dicetak "Hello" sehingga range slicenya adalah x[0:5]

Misalnya pada variabel x akan dicetak "World" sehingga range slicenya adalah x[0:]

```
1   x = "Hello World"
2
3   print (x[0:5])
4   print (x[0:])
```

```
PS C:\Users\Arif Nurrochman\OneDrive\Documents\latih
ocuments/latihanpython_cA23/hello_world.py"
Hello
Hello World
```

BAB 13
Pengertian Operator Logika pada *Python*

Operator logika pada *Python* merupakan operator yang digunakan untuk melakukan komputasi boolean, salah satu tipe data pada bahasa pemrograman *Python*. Sama halnya seperti bahasa pemrograman lainnya, operator logika *Python* merupakan aplikasi dari disiplin ilmu matematika modern.

A. Pengertian Operator Logika pada *Python*

Operator logika pada *Python* adalah operasi logika matematika untuk melakukan operasi komputasi dari data boolean, yang terdiri dari *True* dan *False*. True bernilai benar dan False bernilai salah.

B. Operasi Matematika pada Operasi Logika *Python*

Berikut beberapa operasi matematika yang erat kaitannya dengan operasi logika

Operasi Matematika	Fungsi
<	Lebih kecil dari
<=	Lebih kecil atau sama dengan
>	Lebih besar dari
>=	Lebih besar atau sama dengan
==	Sama dengan
!=	Tidak sama dengan

C. Menggunakan Jenis Operator Logika pada *Python*

Terdapat 4 operator logika yang dapat digunakan pada bahasa pemrograman *Python*.

Operator	*Python* Function	Symbolic Function
AND	and()	&
OR	or()	|
XOR	xor()	^
NOT	not()	

1. Operator AND

Operator *AND* pada *Python* dapat dilakukan menggunakan function *and()* atau **&.** Nilai kebenaran operator *AND* adalah benar ketika kedua operand bernilai benar. Operand adalah nilai yang digunakan pada operator logika. Berikut tabel kebenaran operator *AND*.

AND	*Operand 1*	*Operand 2*
True	True	True
False	True	False
False	False	True
False	False	False

2. Operator OR

Operator *OR* pada *Python* dapat dilakukan menggunakan fungsi *OR()* atau **|**. Operator *OR* mempunyai nilai kebenaran salah saat kedua operand bernilai salah. Berikut tabel kebenaran operator logika *OR*.

OR	Operand 1	Operand 2
True	True	True
True	True	False
True	False	True

OR	Operand 1	Operand 2
False	False	False

3. Operator *XOR*

Operator *XOR* pada *Python* adalah operasi logika dari *OR* Eklusif. Dalam aljabar boolean, Operasi *XOR* mempunyai definisi setiap tapi tidak semua. Operator *XOR* memberikan nilai kebenaran benar (1) saat jumlahan operand adalah ganjil.

XOR	**Operand 1**	**Operand 2**
True (2)	True (1)	True (1)
True (1)	True (1)	False (0)
True (1)	False (0)	True (1)
False (0)	False (0)	False (0)

4. Operator NOT

Operator *NOT* pada *Python* berati negasi dari dari operand yang dimuatnya.

BAB 14
Struktur Data List

Bagaimana caranya menyimpan banyak data dalam satu veriabel dengan menggunakan *List*. *List* adalah struktur data pada *Python* yang mampu menyimpan lebih dari satu data, seperti array.

Pada kesempatan ini, kita akan membahas cara menggunakan *list* di *Python* dari yang paling sederhana sampai yang sedikit kompleks.

Poin-poin yang akan dipelajari list.

1. Cara Membuat List dan Mengisinya
2. Cara Mangambil nilai dari List
3. Cara Menambahkan dan Menghapus isi List
4. Operasi pada List
5. List multi dimensi

A. Cara Membuat List di *Python*

List dapat kita buat seperti membuat variabel biasa, namun nilai variabelnya diisi dengan tanda kurung siku ([]).

Contoh:

```
# Membuat List kosong warna = []

# Membuat list dengan isi 1 item hobi =
["membaca"]
```

Apabila list-nya memiliki lebih dari satu isi, maka kita bisa memisahnya dengan tanda koma.

Contoh:

buah = ["jeruk", "apel", "mangga", "duren"]

Jenis jenis yang diperbolehkan dalam *List*. *list* dapat diisi dengan tipe data apa saja, *string, integer, float, double, boolean, object,* dan sebagainya.

Kita juga bisa kombinasi isinya.

Contoh:

```
laci = ["buku", 21, True, 34.12]
```

Ada empat jenis tipe data pada list laci:

1. "buku" adalah tipe data string;
2. 21 adalah tipe data integer;
3. True adalah tipe data boolean;
4. dan 34.12 adalah tipe data float.

B. Cara Mengambil Nilai dari List

Setelah kita tahu cara membuat dan menyimpan data di dalam *List*, mari kita coba mengambil datanya. List sama seperti array, list juga memiliki nomer indeks untuk mengakses data atau isinya. Nomer indeks *list* selalu dimulai dari nol (0).

Nomer indeks ini yang kita butuhkan untuk mengambil isi (item) dari *list*.

Contoh:

```
# Kita punya list nama-nama buah
hobby = ["memancing", "membaca", "menulis",
"mendengar musik"]

# Misanya kita ingin mengambil mangga
# Maka indeknya adalah 2
print (hobby[2])
```

Akan menghasilkan output sebagai berikut:

"membaca"

Latihan 1: Membuat Program dengan List

Untuk memantapkan pemahaman, silahkan coba latihan berikut.

1. Buat sebuah list untuk menyimpan kenalanmu
2. Isi list sebanyak 5
3. Tampilkan isi list indeks nomer 3
4. Tampilkan semua teman dengan perulangan
5. Tampilkan panjang list

```
# Buat list untuk menampung nama-nama teman
my_friends = ["Anggun", "Dian", "Agung", "Adi", "Adam"]

# Tampilkan isi list my_friends dengan nomer indeks 3
print   ("Isi   my_friends  indeks  ke-3    adalah:{}".format(my_friends[3]))

# Tampilkan semua daftar teman
print   ("Semua teman:  ada {} orang".format(len(my_friends)))
for friend in my_friends:
    print (friend)
```

Pada kode di atas, kita menggunakan fungsi len() untuk mengambil panjang *list*.
Hasil outputnya:
Isi my_friends indeks ke-3 adalah Adi
Semua teman ada 5 orang
Anggun
Dian
Agung
Adi
Adam

1. **Menghapus Item di List**

   Untuk menghapus salah satu isi dari *List*, kita bisa menggunakan perintah del. Perintah del akan menghapus sebuah variabel dari memori.

   Contoh:

```
# Membuat List
todo_list = ["Teknik Informatika", "Sistem Informasi"]
# Misalkan kita ingin menghapus "Belajar Sulap" # yang berada di indeks ke-3
del (todo_list[3])
print (todo_list)
```

   Hasilnya, "`Belajar Sulap`"akan dihapus:
   ['Teknik Informatika', 'Sistem Informasi']

2. **Memotong list**

Seperti string, list juga dapat dipotong-potong. Contoh:

```
# Kita punya list BAHASA
bahasa = ["PHP", "Python", "Javascript", "HTML", "CSS", "C++"]
# Kita potong dari indeks ke-1 sampai ke-5
print (bahasa[1:5])
```

Hasilnya:

```
["PHP", "Python", "Javascript", "HTML", "CSS", "C++"]
```

3. **Operasi List**

Ada beberapa operasi yang bisa dilakukan terhadap List, diantaranya:

- Penggabungan (+)
- Perkalian (*)

Contoh:

```
# Beberapa list lagu
list_lagu = ["No Women, No Cry", "Dear God"]

# playlist lagu favorit
playlist_favorit = ["Break Out", "Now Loading!!!"]

# Mari kita gabungkan keduanya
semua_lagu = (list_lagu + playlist_favorit)

print (semua_lagu)
```

Hasilnya:

['No Women, No Cry', 'Dear God', 'Break Out', 'Now Loading!!!']

Sedangkan untuk operasi perkalian hanya dapat dilakukan dengan bilangan.

Contoh:

```
# playlist lagu favorit
playlist_favorit = ["Break Out", "Now Loading!!!"]
# ulangi sebanyak 5x
ulangi = 5
now_playing = (playlist_favorit * ulangi)
print (now_playing)
```

Hasilnya:
['Break Out', 'Now Loading!!!', 'Break Out', 'Now Loading!!!', 'Break Out', 'Now Loading!!!', 'Break Out', 'Now Loading!!!']

4. List Multi Dimensi

Pada contoh-contoh di atas, kita hanya membuat list satu dimensi saja. List dapat juga memiliki lebih dari satu dimensi atau disebut dengan multi dimensi. List multi dimensi biasanya digunakan untuk menyimpan struktur data yang kompleks seperti tabel, matriks, graph, tree, dan sebagainya.
Contoh:

```
# List minuman dengan 2 dimensi
list_minuman = [
["Kopi", "Susu", "Teh"],
["Jus Apel", "Jus Melon", "Jus Jeruk"],
["Es Kopi", "Es Campur", "Es Teler"]]

# Cara mengakses list multidimensi
# misalkan kita ingin mengambil "es kopi"
print (list_minuman[2][0])
```

Angka 2 pada kode di atas, menunjukan indeks list yang akan kita akses. Kemudian setelah dapat list-nya baru kita ambil isinya.
Hasil outputnya:
"Es Kopi"
Bagaimana kalau kita ingin menampilkan semua isi dalam list multi dimensi, tinggal gunakan perulangan bersarang.

BAB 15
Membaca dan Menulis File di *Python*

Membaca dan menulis file adalah teknik dasar yang harus dipahami dalam pemrograman *Python*, karena banyak digunakan untuk pengolahan dan pemerosesan file.

Paham cara membaca dan menulis file dengan *Python* akan membuatmu mampu membuat aplikasi yang bisa mengambil dan menyimpan data ke file.

Selain itu juga, kamu akan lebih mudah memahami beberapa materi *Python* selanjutnya, seperti membaca dan *parshing* file JSON, XML, CSV, XLS, dan sebagainya.

Ada banyak sekali tipe file pada computer seperti, dokumen, video, gambar, audio, arsip, dll.

Pada *Python*, file hanya dikelompokkan menjadi dua tipe:

1. File Teks: File yang berisi teks. Setiap baris teks memiliki EOL *(End of Line)*.
   Contoh: TXT, MD, CSV, JSON, dsb.
2. File Binary: File yang bukan teks, hanya bisa diproses oleh program tertentu yang memahami strukturnya. Contoh: EXE, JPG, MKV, M4A, 3GP, dsb.

Pada tutorial ini, kita akan hanya belajar cara membaca dan menulis file teks saja.

Untuk file *binary*, mungkin nanti di kesempatan yang lain.

A. Cara Membaca File di *Python*

Python sudah menyediakan fungsi `open()`, untuk membaca dan menulis file.

Fungsi ini memiliki dua parameter, yaitu nama file dan mode.

Objek `file`adalah variabel objek yang menampung isi file. Kita bisa melakukan pemrosesan file berkatnya.

Nama file bisa kita isi langsung apabila file-nya terletak dalam satu direktori dengan skrip *Python*. Namun, apabila terletak di direktori yang berbeda, maka kita harus memberikan alamat *path* file-nya.

Misalnya seperti ini:

```
obj_file = open("/path/ke/file.txt", "r")
```

Kemudian untuk parameter *mode,* fungsinya untuk menentukan hak akses terhadap file.

Ada beberapa mode yang tersedia:

Mode	Ketereangan
"r"	hanya baca saja
"w"	akses untuk menulis file, jika file sudah ada, maka file akan di replace dan diganti dengan yang baru ditulis
"a"	digunakan untuke *append* atau menambah data ke file, artinya jika sudah ada data dalam file, maka akan ditambahkan dan tidak di-replace
"r+"	digunakan untuk membaca sekaligus menulis data ke file

Pada *Windows*, kadang ditambahkan `"b"` dibelakangnya, seperti: `"rb", "wb", "ab", "r+b"`.

Artinya untuk membuka file dalam mode *binary*.

Seperti yang kita ketahui antara EOL Unix dan *Windows* berbeda, kadang file *binary* seperti JPG dan EXE bisa rusak pada *Windows* kalau tidak ditambahkan akhiran `"b"`.

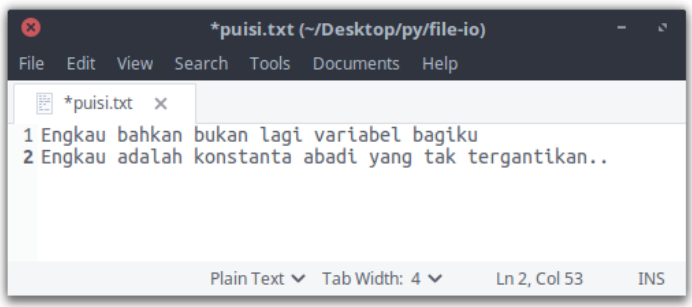
Tapi di Unix/*Linux* tidak masalah, tanpa harus menggunakan akhiran `"b"`.
Sekarang kita latihan praktek dan latihan agar lebih paham. Sekarang silahkan buat direktori bernama `file-io`, lalu di dalamnya buat file `puisi.txt` dan baca`_file.py`.

```
file-io/
├── baca_file.py
└── puisi.txt
```

Setelah itu, silahkan buka file `puisi.txt` dengan notepad atau teks editor, kemudian isi dengan teks berikut:

```
Engkau bahkan bukan lagi variabel bagiku
Engkau adalah konstanta abadi yang tak tergantikan...
```

Jangan lupa di simpan.



B. Membaca File Per Baris

Selanjutnya kita akan mulai menulis kode programnya. Silahkan buka file `baca_file.py` kemudian ketik kode berikut. *(usahakan diketik sendiri dan jangan dicopas)*

```
# buka file
file_puisi = open("puisi.txt", "r")

# baca isi file
print (file_puisi.readlines())

# tutup file
file_puisi.close
()
```

kita membuka file dengan fungsi `open()`, selanjutnya mencoba membaca isinya per baris dengan *method* `readlines()`, dan terakhir menutup file dengan `close()` agar dihapus dalam memori.

Saat dieksekusi, kode di atas akan menghasilkan output berupa *list*, karena kita menggunakan method `readlines()`.

Perhatikan \n adalah EOL *(End of Line)*, simbol ini akan membuat baris baru.

Selanjutnya mari kita coba ubah kodenya menjadi seperti ini:

```
# buka file
file_puisi = open("puisi.txt", "r")

# baca isi file
puisi = (file_puisi.readlines())

# cetak baris
pertama print
(puisi[0])
# cetak baris kedua
print (puisi[1])

# tutup file
file_puisi.close()
```

Hasil outputnya:

Engkau bahkan bukan lagi variabel bagiku Engkau adalah konstanta abadi yang tak tergantikan...

Bagus, kita sudah berhasil membaca dan mencetak isi file. "Tunggu dulu…bagaimana kalau filenya memiliki banyak baris?"

kita tinggal menggunakan pengulangan untuk mencetaknya. Silahkan ubah lagi kodenya menjadi seperti ini:

```
# buka file
file_puisi = open("puisi.txt", "r")

# baca isi file
puisi = file_puisi.readlines()
# cetak isi file dengan perulangan for teks in puisi:
print ("teks")

# tutup file file_puisi.close()
```

Coba tambahkan isi file `puisi.txt`dan coba lihat hasil outputnya sekarang.

C. Membaca Semua Teks dalam File

Kalau tadi kita membaca isi file per baris, sekarang kita coba baca semua teks menggunakan method `read()`.
Silahkan ikuti kode berikut:

```
# buka file
file_puisi = open("puisi.txt", "r")

# baca isi file
puisi = file_puisi.read()
```

```
# cetak isi file print
(puisi)

# tutup file
file_puisi.close()
```

"Apa bedanya method `read()` dengan `readlines()`?"
Bedanya:

Method `read()` membaca seluruh teks dan akan mengembalikan nilai string. Sedangkan `readlines()` membaca isi file per baris dan akan mengembalikan nilai berupa `list`. Perlu diketahui, method `read()` dan `readlines()` hanya sekali pakai. Artinya, hanya diekekusi sekali saja.
Eksekusi pertama, dia akan mengembalikan nilai berdasarkan isi filenya.
Eksekusi kedua, dia akan mengembalikan nilai kosong.
Karena itu, kita harus menyimpan hasilnya ke dalam variabel seperti `puisi`.

Contoh:

```
>>> f = open("puisi.txt","r")
>>> print f.read()
Engkau bahkan bukan lagi variabel bagiku Engkau adalah
konstanta abadi yang tak tergantikan...
Tak ada yang bisa memisahkan kita

>>>
f.read() "
>>>f.readlines() []
```

Lihat:

Pada eksekusi kedua, method `read()` dan `readlines()` mengembalikan nilai kosong.

Selanjutnya kita akan mencoba menulis data ke file.

D. Cara Menulis File di *Python*

Seperti yang sudah kita ketahui, ada tiga mode yang digunakan bila ingin menulis file, yaitu: `"w", "a",` dan `"r+".`

Apa saja perbedaan dari ketiga mode itu. Menulis File dengan Mode "W"

Silahkan buat program baru bernama `tulis_bio.py`, kemudian ikuti kodenya seperti ini.

```
print "Selamat datang di Program Biodata"
print "================================="

# Ambil input dari user
nama = input("Nama: ")
umur = input("Umur: ")
alamat = input("Alamat: ")
```

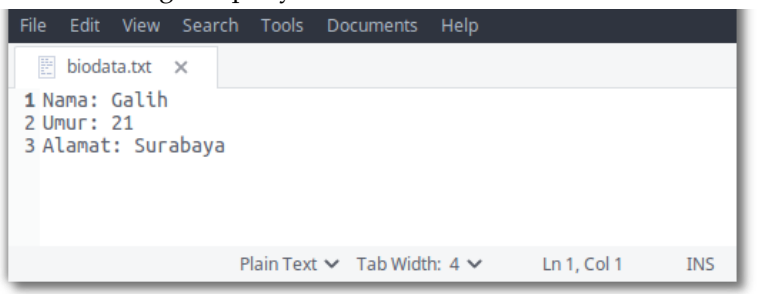
```
# format teks
teks = "Nama: {}\nUmur: {}\nAlamat:
{}".format(nama, umur, alamat)

# buka file untuk ditulis
file_bio = open("biodata.txt", "w")

# tulis teks ke file
file_bio.write(teks)

# tutup file
file_bio.close()
```

Setelah itu, coba eksekusi programnya.
Maka sekarang kita punya file baru bernama `biodata.txt`.



Apa bila file itu sudah ada, maka akan di-replace atau ditulis ulang dengan yang baru.
Ada dua *method* yang bisa kita gunakan untuk menulis file

1. `write()` : parameternya teks (string)
2. `writelines()` : parameternya teks dalam bentuk *list*.

Contoh:

```
teks = "ini sebuah teks"
teks_list ["apel", "mangga", "anggur"]

f = open("file.txt", "w")
f.write(teks)
f.writelines(teks_list)
```

perbedaan Method `write()` akan menulis semua teks, sedangkan `writelines()` akan menulis per baris.

E. Menyisipkan Data ke File

Apabila kita tidak ingin menulis ulang atau menindih file yang sudah ada, kita bisa menggunakan mode `"a"` *(append)* untuk menulisnya.

Silahkan modifikasi program sebelumnya.

```
print "Selamat datang di Program
Biodata" print
"================================="

# Ambil input dari user
nama = input("Nama: ")
umur = input("Umur: ")
alamat = input("Alamat: ")

# format teks
teks = "\nNama: {}\nUmur: {}\nAlamat: {}\n---
".format(nama, umur, alamat)
```

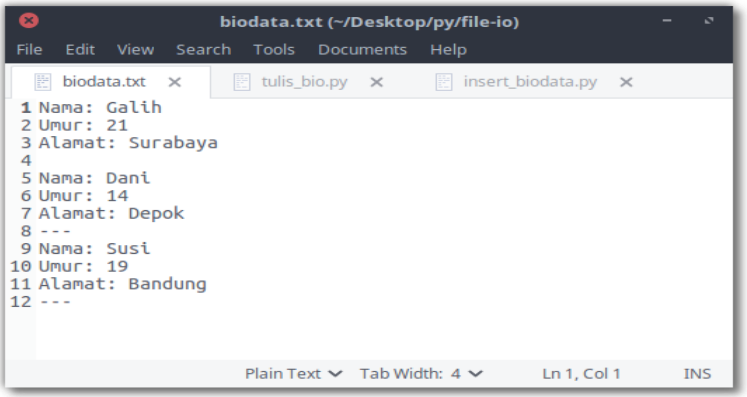
```
# buka file untuk ditulis
file_bio = open("biodata.txt", "a")

# tulis teks ke file
file_bio.write(teks)

# tutup file
file_bio.close()
```

Setelah itu, coba eksekusi dan masukan beberapa data.

Maka sekarang di file `bioadta.txt`ada data baru.



F. Membaca dan Menulis File dengan Mode "r+"

Saat kita ingin membaca dan menulis file bersamaan, kita bisa saja membuat dua objek file seperti ini:

```
file_baca = open("nama_file.txt", "r")
file_tulis = open("nama_file.txt", "w")
```

Namun, sepertinya akan memakan banyak memori, karen kita membuat banyak objek. Maka cara terbaik yang digunakan adalah menggunakan mode *"r+"*membaca sekaligus menulis. Silahkan modifikasi kembali program yang tadi.

```
print "Selamat datang di Program Biodata"
print "================================="

# buka file untuk dibaca dan ditulis
file_bio = open("biodata.txt", "r+")

teks = file_bio.read()

# cetak isi file
print teks

# Ambil input dari user
nama = input("Nama: ") umur
= input("Umur: ")
alamat = input("Alamat: ")

# format teks
teks = "\nNama: {}\nUmur: {}\nAlamat: {}\n---
".format(nama, umur, alamat)

# tulis teks ke file
file_bio.write(teks)

# tutup file
file_bio.close()
```

Setelah itu, coba eksekusi programnya.

Maka sekarang akan ada data baru yang ditambahkan.

Sifat dari mode `"r+"`sama seperti `"a"`, dia tidak menindih data yang sudah ada.

G. Menggunakan with dan as

Pada contoh-contoh di atas, kita selalu menutup file dengan method `close()`.

Sebenarnya file bisa ditutup otomatis dengan mengguakan `with`. Contoh:

```
with open("dokumen.txt", "r") as dok:
    print dok.read()
```

Setelah blok `with` dieksekusi, file akan ditutup dan otomatis dihapus dari memori.

Menggunakan *Exception*

Ada kalanya saat kita baca file, tapi filenya belum ada.

Maka biasanya akan terjadi `IOerror`.

```
IOError: [Errno 2] No such file or directory:
'file.txt'
```

Hal ini, bisa kita atasi dengan menggunakan *exception*. Contoh:

```
try:
    f = open("file.txt","r")
except IOError as err:
    print "Terjadi kesalahan: {}".format(err)
```

Maka, bila error…akan menghasilkan output:

```
Terjadi kesalahan: [Errno 2] No such file or
directory: 'file.txt'
```

BAB 16
Fungsi dan Prosedur pada *Python*

Pada pembuatan program yang kompleks dan memiliki banyak fitur, kita diharuskan menggunakan fungsi. Bisa jadi kita akan kerepotan menulis kode programnya, karena banyak yang harus ditulis dan kode akan menjadi sulit dibaca dan dirawat *(maintenance)*.

Dengan fungsi, kita dapat memecah program besar menjadi sub program yang lebih sederhana. Masing-masing fitur pada program dapat kita buat dalam satu fungsi. Pada saat kita membutuhkan fitur tersebut, kita tinggal panggil fungsinya saja. Hal ini akan kita coba pada contoh program yang sudah saya sediakan di bawah ini.

Terlebih dahulu kita memahami teori dasar dan hal apa saja yang harus kita ketahui tentang fungsi di *Python*.

A. Cara Membuat Fungsi pada *Python*

Fungsi pada *Python*, dibuat dengan kata kunci *def* kemudian diikuti dengan nama fungsinya.

Contoh:

```
def nama_fungsi():
    print ("Hello Mahasiswa Global")
```

Sama seperti blok kode yang lain, kita juga harus memberikan identasi (tab atau spasi 2x) untuk menuliskan isi fungsi.

Setelah dibuat fungsinya, lalu kita dapat memanggilnya kembali seperti dibawah ini.

nama_fungsi()

Sebagai contoh, coba tulis kode program berikut:

```
# Membuat
Fungsi def
salam():
    print ("Hello, Selamat Pagi")

## Pemanggilan Fungsi
salam()
```

Hasilnya:
Hello, Selamat Pagi
Coba panggil sebanyak 3x:

```
# Membuat
Fungsi def
salam():
     print ("Hello, Selamat Pagi")

## Pemanggilan Fungsi
salam()
salam()
salam()
```

Hasilnya:
Hello, Selamat Pagi
Hello, Selamat Pagi
Hello, Selamat Pagi

Intinya apapun yang ada di dalam fungsi, ketika dipanggil itulah yang akan dilakukan.
FYI: fungsi juga dapat dipanggil pada fungsi lain, bahkan bisa memanggil dirinya sendiri. Fungsi yang memanggil dirinya sendiri, disebut fungsi rekursif.

B. Fungsi dengan Parameter

Bagaimana kalau kita ingin memberikan input nilai ke dalam fungsi, kita bisa manfaatkan parameter.

Parameter adalah variabel yang menampung nilai untuk diproses di dalam fungsi.
Contoh:

```
def salam(ucapan):
     print(ucapan)
```

Pada contoh diatas, untuk membuat fungsi dengan parameter ucapan. Dan bagaimana cara memanggilnya.
Cara pemanggilan fungsi yang memiliki parameter adalah seperti dibawah ini:
salam("Selamat siang")
"Selamat siang" adalah nilai parameter yang kita berikan.

Lalu bagaimana kalau parameternya lebih dari satu, Kita bisa menggunakan tanda koma (,) untuk memisahnya.
Contoh:

```
# Membuat fungsi dengan parameter
def luas_lingkaran(jari, jari2):
    luas = (jari * jari2) * 3.14
    print ("Luas lingkaran: %d" % luas)

# Pemanggilan fungsi
luas_lingkaran(4, 4)
```

Hasilnya:
Luas segitiga: 50

C. Fungsi yang Mengembalikan Nilai

Fungsi yang tidak mengembalikan nilai biasanya disebut dengan prosedur. Namun, kadang kita membutuhkan hasil proses dari fungsi untuk digunakan pada proses berikutnya.

Maka fungsi harus mengembalikan nilai dari hasil pemrosesannya.

Cara mengembalikan nilai adalah dengan menggunakan kata kunci *return* lalu diikuti dengan nilai atau variabel yang akan dikembalikan.
Contoh:

```
1  def luas_persegi(sisi):
2      luas = sisi * sisi
3      return luas
4
5  # pemanggilan fungsi
6  print ("Luas persegi: %d" % luas_persegi(6))
```

Hasilnya:
Luas persegi: 36
Apa bedanya dengan fungsi *luas_segitiga()* yang tadi, fungsi *luas_segitiga()* kita melakukan print dari hasil pemrosesan secara langsung di dalam fungsinya. Sedangkan fungsi

luas_persegi(), kita melakukan print pada saat pemanggilannya. Jadi, fungsi *luas_persegi()* akan bernilai sesuai dengan hasil yang dikembalikan. Sehingga kita dapat memanfaatkannya untuk pemrosesan berikutnya.

D. Variabel Global dan Lokal pada *Python*

Saat kita menggunakan fungsi, maka kita mengetahui yang namanya variabel Global dan Lokal. Variabel Global adalah variabel yang bisa diakses dari semua fungsi, sedangkan variabel lokal hanya bisa diakses di dalam fungsi tempat ia berada saja.

Pada *Python*, urutan pengaksesan variabel *(scope)* dikenal dengan sebutan LGB (Local, Global, dan Build-in).

Jadi program *Python* mulai mencari vairabel lokal terlebih dahulu, kalau ada maka itu yang digunakan.

Tapi kalau tidak ada, pencarian terus ke Global, dan Build-in. Variabel Build-in adalah variabel yang sudah ada di dalam *Python*.

Contoh program:

```
# membuat variabel global
nama = "Selamat"
versi = "1.0.0"

def help():
     # ini variabel lokal
     nama = "Glo"
     versi = "1.0.2"
     # mengakses variabel lokal
     print "Nama: %s" % nama
     print "Versi: %s" % versi

# mengakses variabel global
print "Nama: %s" % nama
print "Versi: %s" % versi

# memanggil fungsi help()
help()
```

Hasilnya:
Nama: Selamat
Versi: 1.0.0
Nama: Glo
Versi: 1.0.2

Perhatikanlah variabel nama yang berada di dalam fungsi *help()* dan diluar fungsi `*help()*. Variabel nama yang berada di dalam fungsi *help()* adalah variabel lokal.

Jadi, saat kita memanggil fungsi *help()* maka nilai yang akan tampil adalah nilai yang ada di dalam fungsi *help()*.

Python mulai mencari dari lokal, ke global, dan build-in.

Kalau di tiga tempat itu tidak ditemukan, maka biasanya akan terjadi *NameError* atau variabel tidak ditemukan.

E. Contoh Program dengan Fungsi

Sekarang tiba saatnya kita membuat program dengan membuat file baru bernama program_*fungsi.py*.

```
def counter_admin():
    counter = input('Ingin menginput lagi ? : ')
    if counter == 'yes':
        print('=' * 20)
        print('Silahkan input data buku selanjutnya... \n')
        Data_Buku()
    elif counter == 'no':
        print('=' * 20)
        print('Silahkan input data identitas anda!')
        Data_Anggota()
    elif counter == 'exit':
        print('Terimakasih!!! Silahkan Berkunjung Kembali Untuk Info Buku Terbaru')
        exit()

    else:
        print('Maaf, Sistem Tidak Mengerti')

```

```
def Data_Buku():

    kode_buku = str(input('Kode Buku : '))
    judul = str(input('Judul Buku : '))
    pengarang = str(input('Pengarang Buku : '))
    penerbit = str(input('Penerbit Buku : '))
    kategori = str(input('Kategori Buku : '))
    tahun_terbit = int(input('Tahun Terbit : '))

    print('=' * 20)
    print('Data buku yang dipinjam  : ')

    print(kode_buku)
    print(judul)
    print(pengarang)
    print(penerbit)
    print(kategori)
    print(tahun_terbit)

    print('=' * 20)
```

```
39      counter_admin()
40
41  def Data_Anggota():
42
43      no_anggota = int(input('No Anggota : '))
44      nama = input('Nama Anggota : ')
45      alamat = input('Alamat : ')
46      telp = int(input('No Telepon : '))
47      kode = input('Kode Keanggotaan : ')
48
49      print('=' * 20)
50      print('Data Member : ')
51
52      print(no_anggota)
53      print(nama)
54      print(alamat)
55      print(telp)
56      if kode == 'A':
57          print('Status : ANGGOTA')
58          print('Biaya : 10000' )
```

```
59      else :
60          print('Status : UMUM')
61          print('Biaya : 50000')
62
63  def Peminjaman():
64
65      from datetime import datetime, timedelta
66      sekarang = datetime.now()
67      tgl_pinjam = print('Tgl Pinjam :',sekarang)
68      no_anggota = input('No. Anggota : ')
69      kode_buku = input('Kode Buku : ')
70
71      print('=' * 20)
72      print('Data Peminjaman : ')
73
74      print('Tgl Pinjam :',sekarang)
75      print('No. Anggota : ',no_anggota)
76      if no_anggota == 'A001':
```

```
77          print('Nama : Iwan')
78      elif no_anggota == 'A002':
79          print('Nama : Budi')
80      else :
81          print('Nama : Nina')
82      print('Kode Buku : ', kode_buku)
83      if kode_buku == 'PK001':
84          print('Judul : P.WEB')
85      elif kode_buku == 'PK002':
86          print('Judul : SINCAN')
87      else :
88          print('Judul : DILAN')
89
90      tgl_kembali = sekarang + timedelta(days=7)
91      print('Tanggal Kembali: ',tgl_kembali)
92  def     main_menu():
93      print('=' * 20)
94      print('PERPUSTAKAAN SMA SYAMMIL')
95      print('Welcome!!!')
96      print('=' * 20)
```

```
97      print('1. Data Buku')
98      print('2. Data Anggota')
99      print('3. Peminjaman')
100     print('4. Pengembalian')
101     print('EXIT')
102     print('Choice : [1/2/3/4/Exit] : ')
103
104     pilihan = input('Your Choice: ')
105
106     if pilihan == '1':
107         print("=" * 20)
108         print("PENDATAAN BUKU")
109         print("=" * 20)
110         Data_Buku()
111     elif pilihan == '2':
112         print("=" * 20)
113         print("PENDATAAN ANGGOTA")
114         print("=" * 20)
115         Data_Anggota()
116     elif pilihan == '3':
```

```
117             Peminjaman()
118         elif pilihan == '4':
119             Pengembalian()
120         else:
121             print('Exit')
122             Exit()
123 
124     def get_login():
125         print('=' * 20)
126         print('Login Member Perpustakaan')
127         print('=' * 20)
128         username = input('Username : ')
129         password = input('Password : ')
130 
131         if username == 'admin' and password == '12345':
132             print('loading...\n\n')
133             print('=' * 20)
134             print('Welcome to Library!!!')
135 
136             main_menu()
```

```
137 
138         else:
139             print('login failed, please try again...')
140 
141     get_login()
142 
143 
```

DAFTAR PUSTAKA

Bini, Ola (2007). Practical JRuby on Rails Web 2.0 Projects: bringing Ruby on Rails to the Java platform. Berkeley: APress. hlm. 3. ISBN 978-1-59059-881-8.

Kuchling, Andrew M. (22 December 2006). "Interview with Guido van Rossum (July 1998)".

Kuchling, A. M. "Functional Programming HOWTO". Python v2.7.2 documentation. Python Software Foundation. tanggal 9 February 2012.

Rossum van, Guido (1993). "An Introduction to Python for NIX/C Programmers". Proceedings of the NLUUG najaarsconferentie (Dutch UNIX users group). even though the design of C is far from ideal, its influence on Python is considerable.

Rossum van, Guido, 2012. Python Software Foundation.Diakses tanggal 20 February 2012. It is a mixture of the class mechanisms found in C++ and Modula-3

Simionato, Michele. "The Python 2.3 Method Resolution Order". Python Software Foundation. The C3 method itself has nothing to do with Python, since it was invented by people working on Dylan and it is described in a paper intended for lispers

Schemenauer, Neil; Peters, Tim; Hetland, Magnus Lie (18 May 2001). "PEP 255 – Simple Generators". Python Enhancement Proposals. Python Software Foundation.

Smith, Kevin D.; Jewett, Jim J.; Montanaro, Skip; Baxter, Anthony (2 September 2004). "PEP 318 – Decorators for Functions and Methods". Python Enhancement Proposals. Python Software Foundation.

https://docs.*Python*-guide.org/starting/install3/*Linux*/

https://www.ics.uci.edu/~pattis/common/handouts/*Python*eclipsejava/*Python*.html

https://docs.*Python*-guide.org/starting/install/win/

https://id.wikipedia.org/wiki/*Python*_(bahasa_pemrograman)

https://www.globalinstitute.com/tutorial/python/

https://belajar*Python*.com/

https://www.advernesia.com/blog/*Python*/variabel-pada-*Python*-dengan-tipe-data-numerik/

https://www.python.org https://docs.*Python*.org https://pypi.org
https://code.activestate.com/recipes/langs/*Python*/
http://www.pyvideo.org
https://scipy.org