Programming for Science

Algoritma & Pemrograman : Pengantar

Aryo Nur Utomo, ST, M.Kom

Aryo Nur Utomo, ST, M.Kom

Under Graduate Electrical Engineering – Computer Engineering, University of Indonesia

Post Graduate Program, Faculty of Computer Science, University of Indonesia

No. HP: 08121961111

Email : aryo.nurutomo@istn.ac.id

Tujuan Umum Mata Kuliah

Mahasiswa dapat menggunakan konsep-konsep algoritma pemrograman untuk membuat program-program skala kecil dengan menggunakan bahasa pemrograman

Materi Perkuliahan

1. Pengenalan Algoritma
2. Pengenalan Bahasa Pemrograman Python
3. Menggunakan Python
4. Menerima Input User
5. Tipe Data dan Operator Python
6. Membuat Pemilihan Kondisi
7. Menyusun Pengulangan
8. Struktur Data Tingkat Lanjut

Materi Perkuliahan

9. Membuat Fungsi Program
10. Penanganan Error (*exception*)
11. Mengenal *Exception* dan *With*
12. Membuat File
13. Pengenalan Class dan Modul
14. Class lanjut
15. Contoh Kasus Python
16. Manipulasi Matriks Dengan Python

Strategi Perkuliahan

- Ceramah
- Diskusi
- Latihan
- Praktik (demonstrasi)

Referensi

- Python Software Foundation Team Python v3.7 Documentation 2001-2020. The Python Software Foundation.
- Downye, Allen, Jeffrey Elkner, dan Chris Meyers, How to Think Like a Computer Scientist: Learning with Python 2002. Green Tea Press: Wellesley, Massachussets
- Swaroop, A Byte of Python, 2005, IonLab: Bangalore, India
- Budi Raharjo, Kumpulan Splusi Pemrograman Python, 2016, Informatika Bandung

Tata Tertib Perkuliahan

- Keterlambatan maksimal: 15 menit.
- Tidak diperbolehkan memakai sandal.
- Berpakaian yang rapi dan sopan.
- Handphone dimatikan atau diset tidak bersuara.

Lain-lain

- Keterlambatan pengumpulan tugas menyebabkan nilai maksimal tugas diturunkan (nilai maksimal = 60) □ penalti per hari = 2,5%
- Tidak ada ujian perbaikan ataupun tambahan tugas untuk memberikan nilai tambahan (1 kali)
- Bila karena sesuatu hal dosen tidak dapat datang sesuai dengan jadwal kuliah, silahkan cek pengumuman di program studi untuk melihat tugas yang diberikan sebagai pengganti perkuliahan atau jadwal pengganti perkuliahan.
- Konsultasi dapat dilakukan via email (sertakan Nama, NPM, Mata Kuliah, Kelas dalam isi email)
- Handout (bentuk PDF) dapat di-download (via koordinator kelas)
- Baca materi perkuliahan sebelum perkuliahan dimulai
- Weblog dosen pengajar akan diinformasikan lebih lanjut.

Programming for Science

Algoritma & Pemrograman : Pengantar

Aryo Nur Utomo, ST, M.Kom

Under Graduate Electrical Engineering – Computer Engineering, University of Indonesia

Post Graduate Program, Faculty of Computer Science, University of Indonesia

No. HP: 08121961111

Email : aryo.nurutomo@istn.ac.id

Apa Itu Algoritma ?

- Definisi :
  - Urutan langkah-langkah untuk memecahkan masalah.
  - Kamus Besar Bahasa Indonesia:
    Algoritma adalah urutan logis pengambilan putusan untuk pemecahan masalah.
- Algoritma dibutuhkan untuk memerintah komputer mengambil langkah-langkah tertentu dalam menyelesaikan masalah.

Penulisan Algoritma

- Dalam bahasa natural (Bahasa Indonesia, Bahasa Inggris, dan bahasa manusia lainnya)
  - Tapi sering membingungkan (ambiguous).
- Menggunakan flow chart (diagram alir)
  - Bagus secara visual akan tetapi repot kalau algoritmanya panjang.
- Menggunakan pseudo-code
  - Sudah lebih dekat ke bahasa pemrograman, namun sulit dimengerti oleh orang yang tidak mengerti pemrograman.

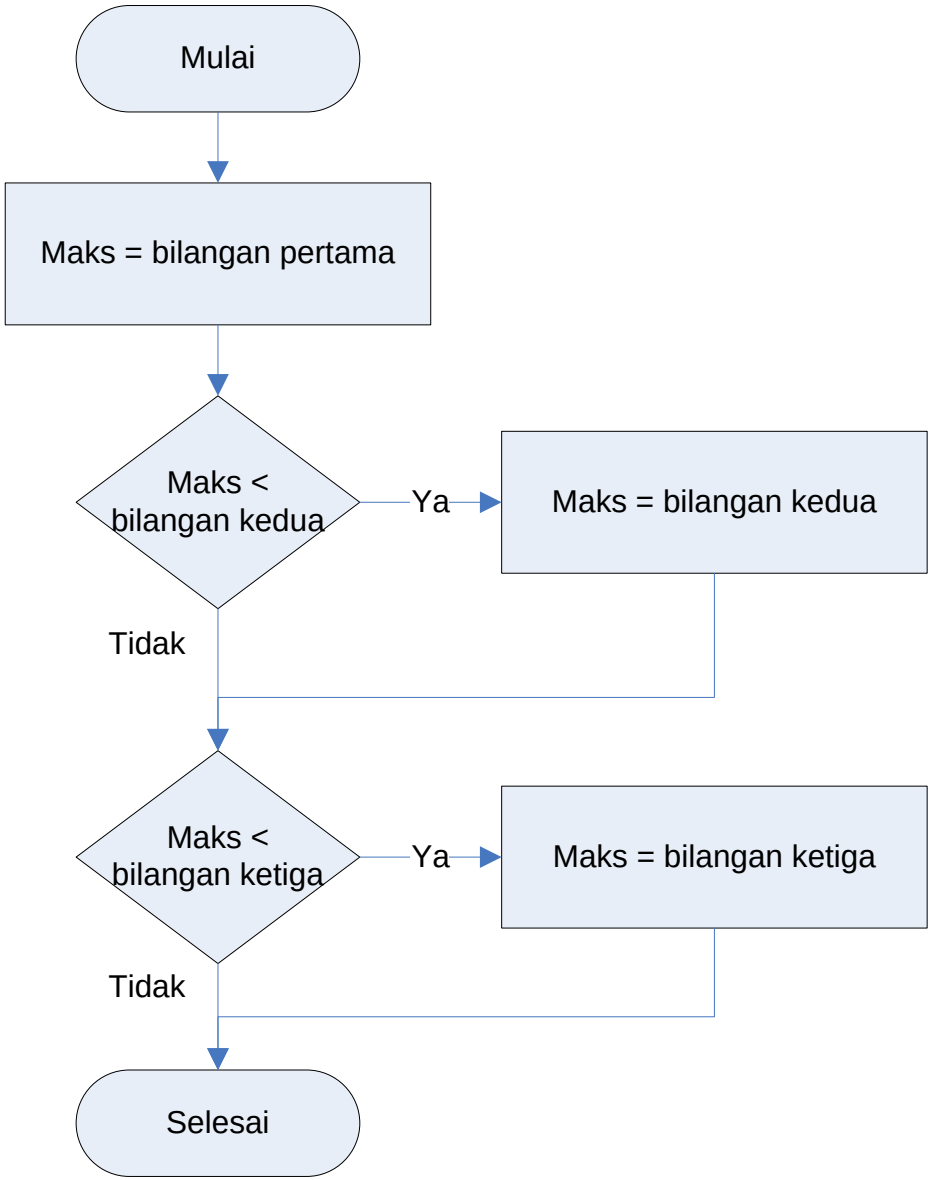
Contoh

- Requirement
  - Buat sebuah algoritma untuk memilih bilangan terbesar dari 3 buah bilangan.
  - Nantinya ini bisa digeneralisir menjadi n buah bilangan

Algoritma Dalam Bahasa Natural

1. Ambil bilangan pertama dan set maks sama dengan bilangan pertama.
2. Ambil bilangan kedua dan bandingkan dengan maks.
3. Apabila bilangan kedua lebih besar dari maks, set maks sama dengan bilangan kedua.
4. Ambil bilangan ketiga dan bandingan dengan maks.
5. Apabila bilangan ketiga lebih besar dari maks, set maks sama dengan bilangan ketiga.
6. Variabel maks berisi bilangan terbesar. Tayangkan hasilnya.

Algoritma Dalam Flowchart



Algoritma Dalam *pseudo-code*

maks ← bilangan pertama

if (*maks* < bilangan kedua)

    maks ← bilangan kedua

if (*maks* < bilangan ketiga)

    maks ← bilangan ketiga

Algoritma Dalam *pseudo-code*

Problem: mencari bilangan terbesar dari dua bilangan yang diinputkan

- Contoh Algoritma:
  1. Masukkan bilangan pertama
  2. Masukkan bilangan kedua
  3. Jika bilangan pertama > bilangan kedua maka kerjakan langkah 4, jika tidak, kerjakan langkah 5
  4. Tampilkan bilangan pertama
  5. Tampilkan bilangan kedua

- Contoh Pseudo-code:
  1. Input a
  2. Input b
  3. If a > b then kerjakan langkah 4 else langkah 5
  4. print a
  5. print b

Flowchart

- Definisi :
  - Bentuk gambar/diagram yang mempunyai aliran satu atau dua arah secara sekuensial.
- Kegunaan :
  - Untuk mendesain program.
  - Untuk merepresentasikan program.
- Maka, *flowchart* harus dapat merepresentasikan komponen-komponen dalam bahasa pemrograman.

Pembuatan Flowchart

- Sebelum pembuatan program
  - Mempermudah programmer dalam menentukan alur logika program.
- Sesudah pembuatan program
  - Menjelaskan alur program kepada orang lain.

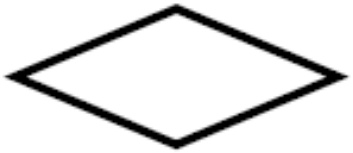
Flowchart

- Secara garis besar, unsur-unsur pemrograman adalah **Input → Proses → Output**. Semua bahasa pemrograman, pasti mempunyai komponen-komponen sebagai berikut :
  - Mendefinisikan variabel (String nama, int a)
  - Input (read), memasukkan nilai ke variabel (nama = “aryo”, bilangan = 3)
  - Percabangan (if, switch-case)
  - Perulangan (while, for, for each, loop)
  - Komputasi (x : + - >= < == ! Shift ,danlainlain)
  - Output (print)

Lambang Flowchart

Keterangan	Lambang
Mulai/selesai *(terminator)*	
Aliran data	
Input/Output	
Proses	

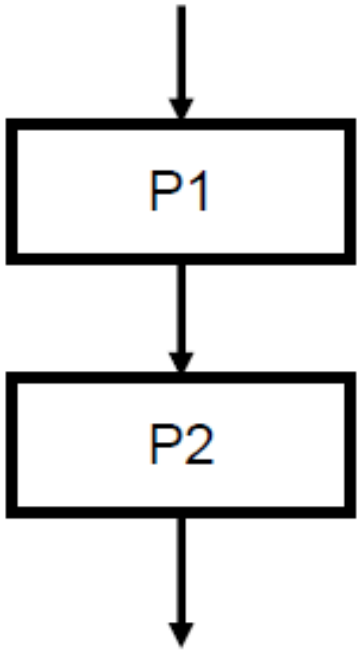
Lambang Flowchart (2)

Keterangan	Lambang
Percabangan *(Decision)*	
Pemberian nilai awal suatu variabel *(Preparation)*	
Memanggil prosedur/fungsi *(Call)*	

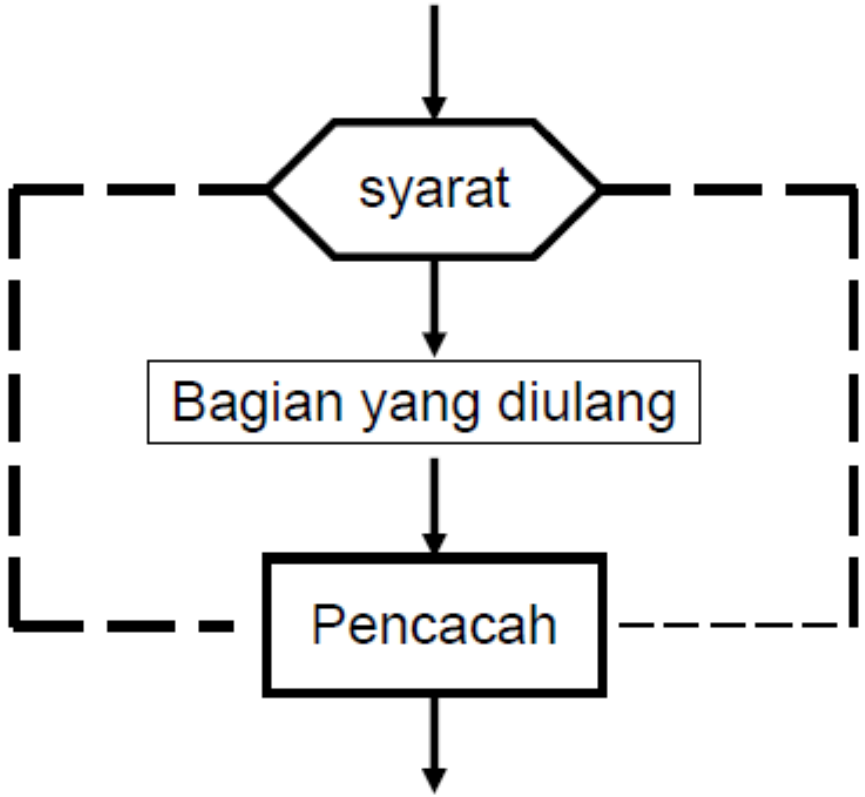
Lambang Flowchart (3)

Keterangan	Lambang
Connector (di halaman yg sama)	
Off page Connector (halaman lain)	
Dokumen / Multi dokumen	
Harddisk	

Lambang Flowchart (4)

Keterangan	Lambang
Sequence Process	

Lambang Flowchart (5)

Keterangan	Lambang
Perulangan	

Contoh Flowchart

Problem :

Menghitung luas persegi panjang

Algoritma :

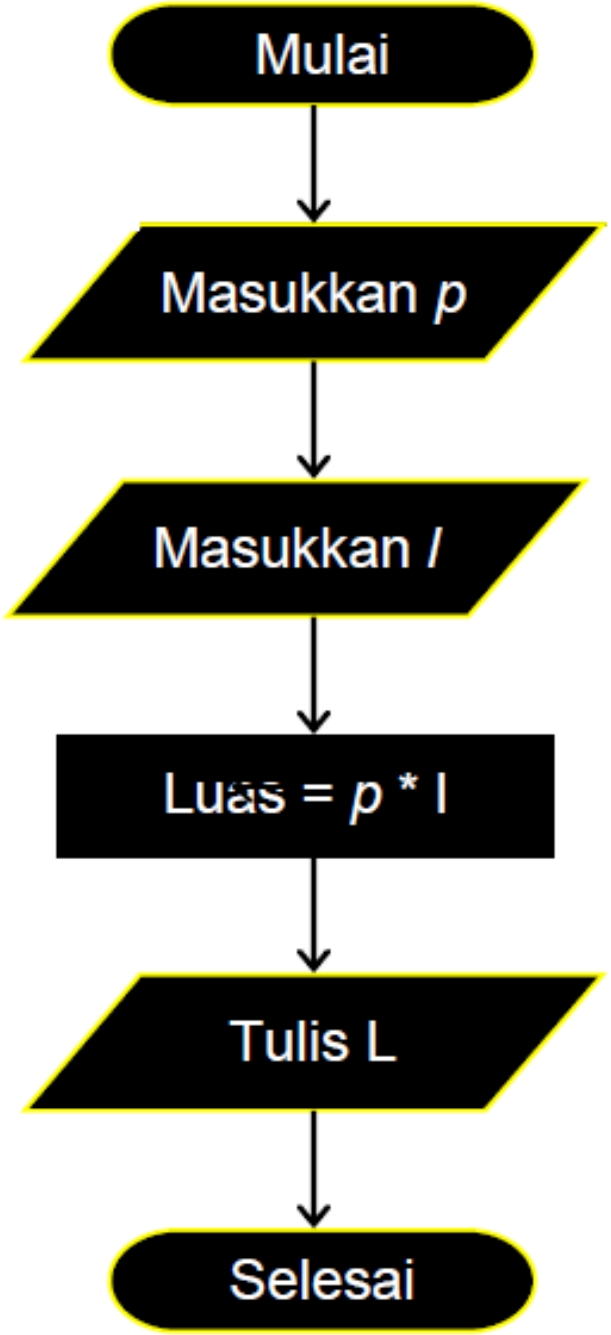
Masukkan panjang (*p*)

Masukkan lebar (*l*)

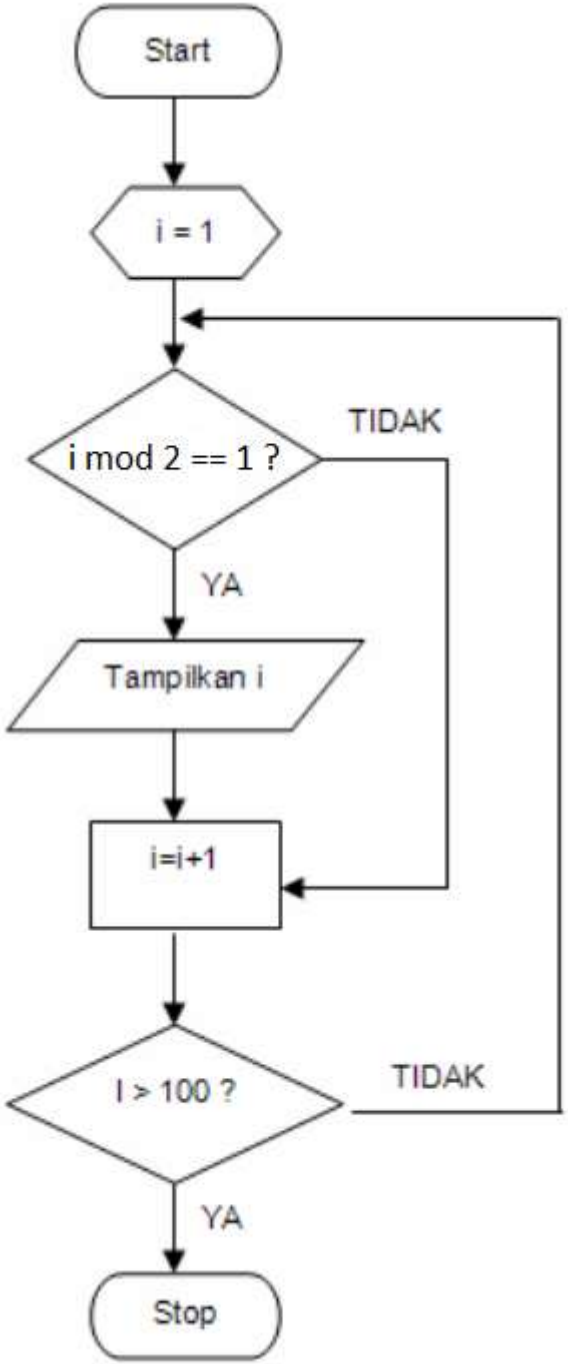
Hitung Luas (L), yaitu

panjang x lebar

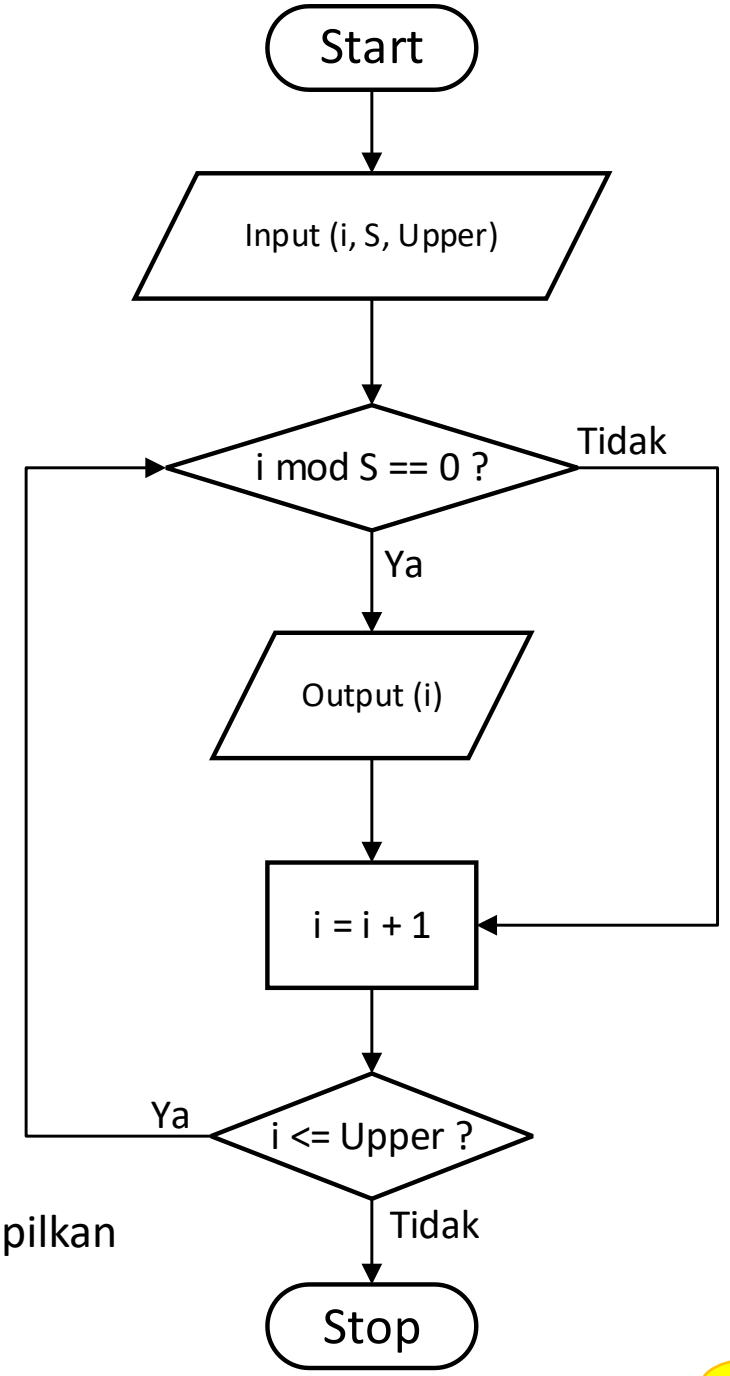
Cetak Luas (L)



Flowchart Bilangan Ganjil 1 s.d 10



Flowchart Kelipatan Bilangan



Keterangan :

S : kelipatan yang ingin ditampilkan

Upper : batas range bilangan

i : bilangan awal

Aspek Penting dari Algoritma

1. *Finiteness*

   Algoritma harus berhenti *after a finite number of steps*

2. *Definiteness*

   Setiap langkah harus didefinisikan secara tepat, tidak boleh membingungkan (*ambiguous*)

3. *Input*

   Sebuah algoritma memiliki nol atau lebih input yang diberikan kepada algoritma sebelum dijalankan

4. *Output*

   Sebuah algoritma memiliki satu atau lebih output, yang biasanya bergantung kepada input

5. *Effectiveness*

   Setiap algoritma diharapkan miliki sifat efektif

Bahasa Pemrograman

Programming Language	Platform Aplikasi		
	Desktop / Server	Web	Mobile
Java JEE, JSE, JME (BlackBerry, Symbian, Java mobile), Android, JSF, JSP, JavaScript	✓	✓	✓
Python	✓	✓	
C, C++	✓		
Visual Basic ASP .Net , C# (Windows)	✓	✓	
Visual Basic (Windows)	✓		
PHP		✓	
Objective C (Apple IOS)	✓		✓
HTML		✓	
Delphi, Pascal	✓		
Matlab	✓		

Lembaga riset programming language : https://www.tiobe.com/tiobe-index/ menempatkan Python dan Java teratas.

Thank You

Programming for Science

Lingkungan Pemrograman Python

Aryo Nur Utomo, ST, M.Kom

Under Graduate Electrical Engineering – Computer Engineering, University of Indonesia

Post Graduate Program, Faculty of Computer Science, University of Indonesia

No. HP: 08121961111

Lingkungan Pemrograman

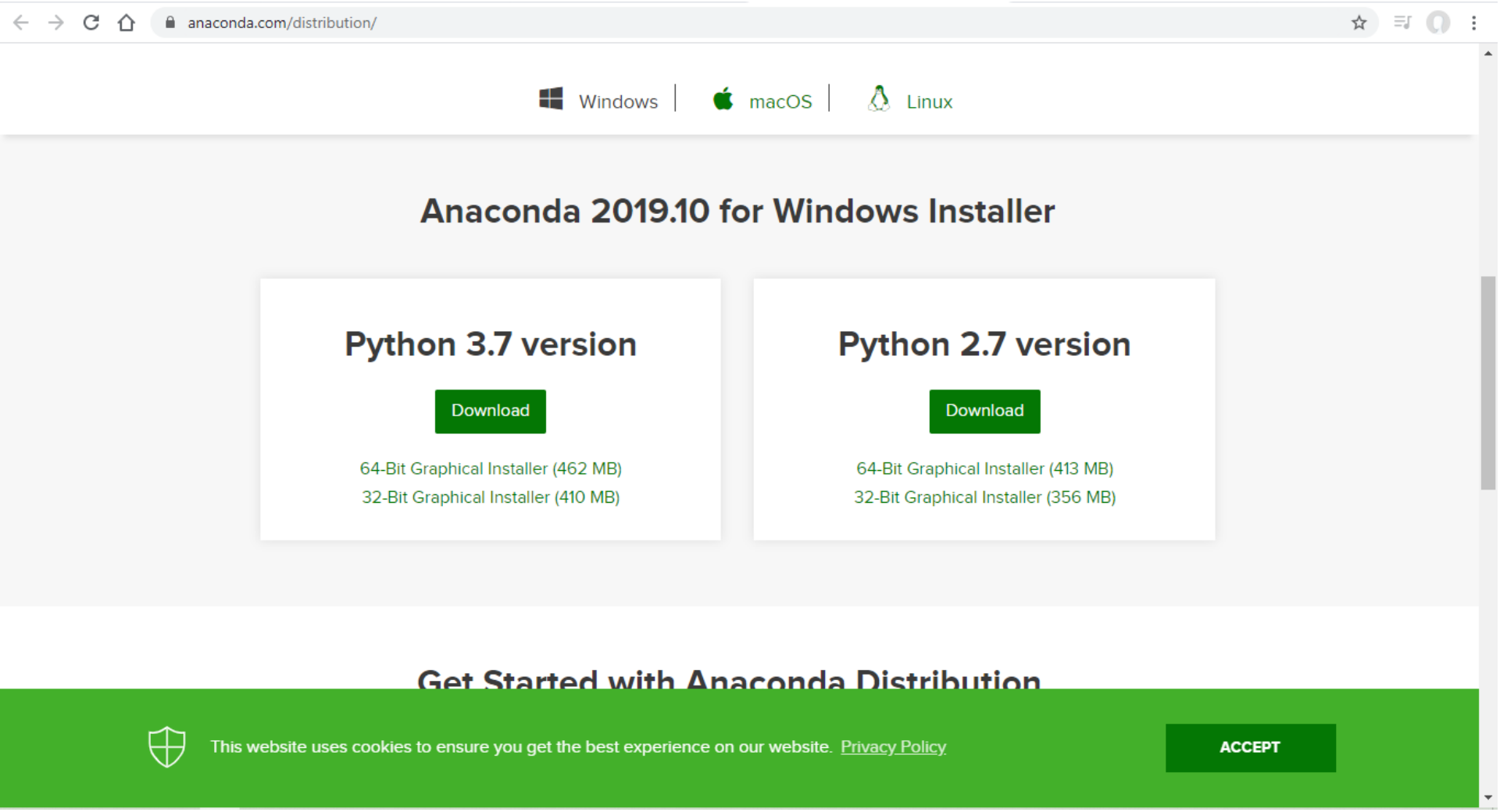
- Win 9x/NT/2000.
- Unix (Solaris, Linux, FreeBSD, AIX, HP/UX, SunOS, IRIX, dll).
- Macintosh (Intel, PPC, 68K).
- OS/2
- DOS
- PalmOS
- Nokia mobile phones
- Windows CE
- Acorn/RISC OS
- BeOS
- Amiga
- VMS/OpenVMS
- QNX
- VxWorks
- Psion
- Raspberry Pi OS
- Pyhton juga dirancang ulang di Java dan .NET Virtual Machine

Material Pemrograman Python

- Distribusi python → menggungakan Anaconda.
- Editor kode → visual studio code.

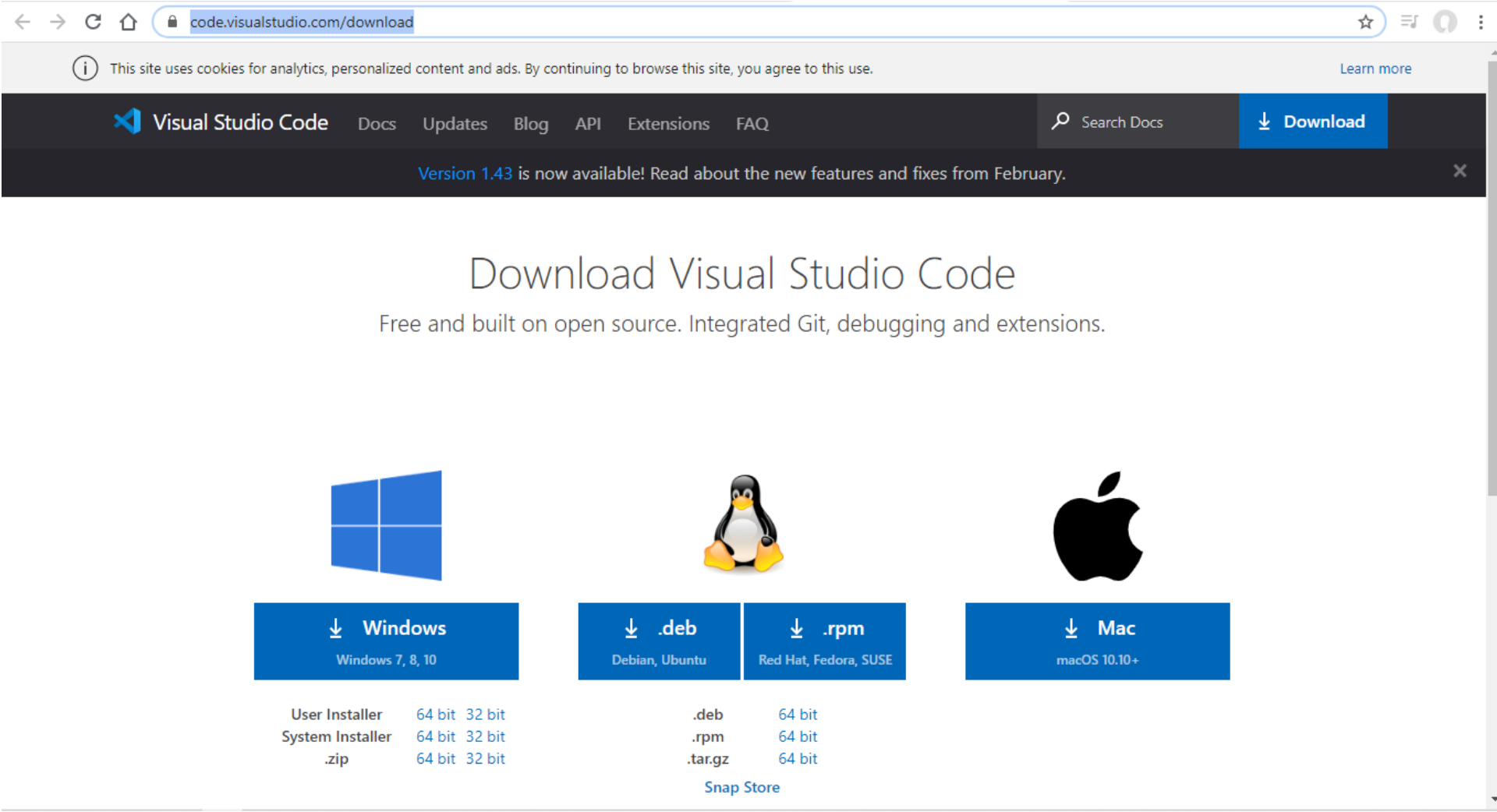
Download Anaconda

https://www.anaconda.com/distribution/



Download Visual Studio Code

https://code.visualstudio.com/download



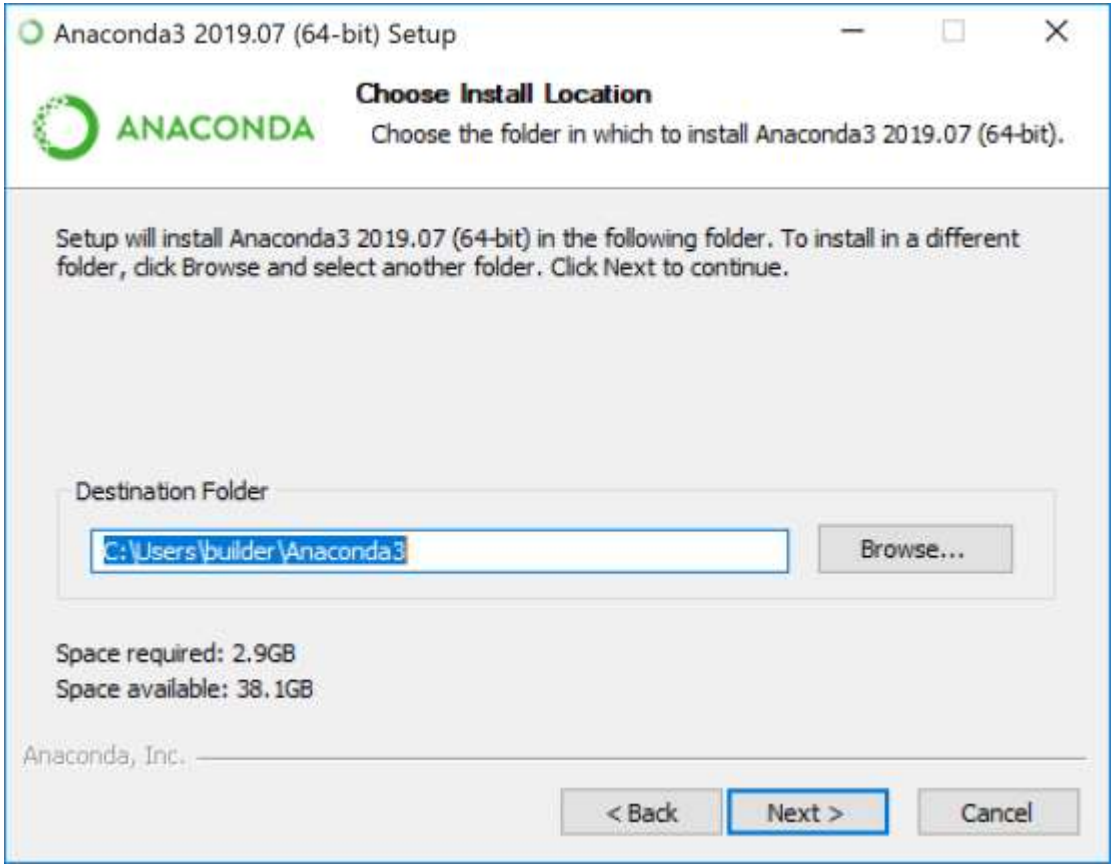
Install Anaconda di Windows

Sumber: https://docs.anaconda.com/anaconda/install/windows/

1. Download anaconda installer.
2. Rekomendasi: cek data integrity dg SHA-256 file download untuk memastikan keaslian file.
3. Double klik aplikasi installer untuk menjalankan.
4. Klik Next
5. Baca ketentuan lisensi dan klik “I Agree”.
6. Pilih install for “Just Me” kecuali anda menginstall untuk semua user (dimana memerlukan akses Windows Administrator) dan klik Next.

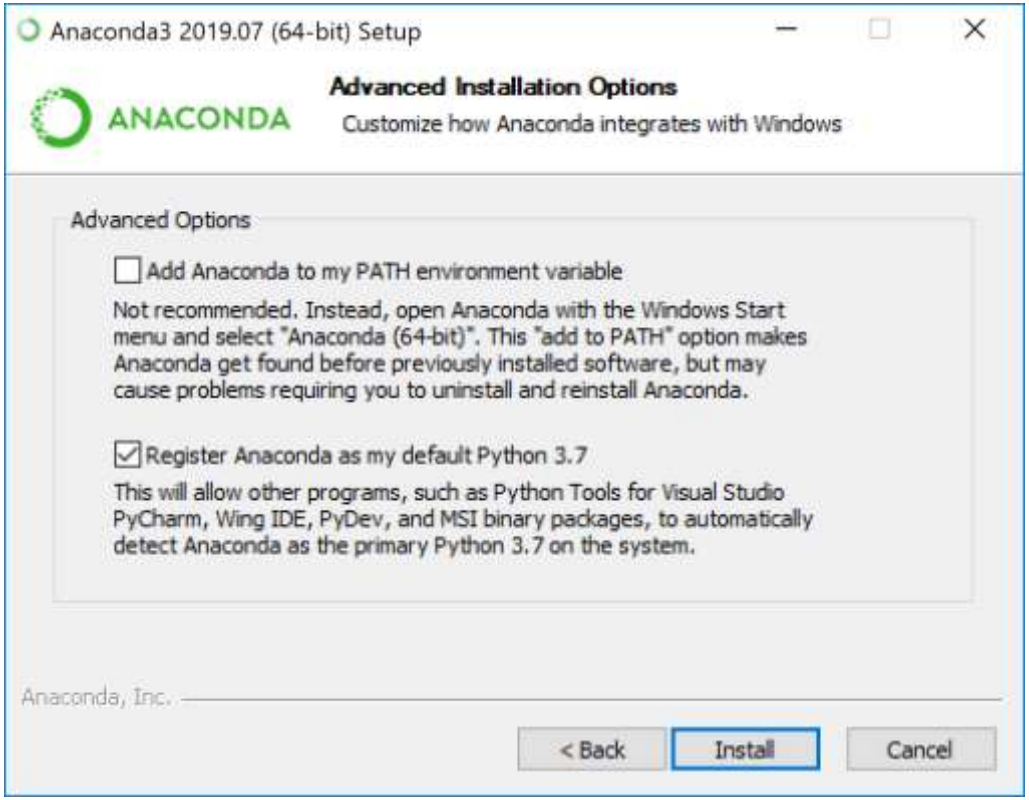
Install Anaconda di Windows

7. Pilih folder tujuan untuk tempat instalasi Anaconda dan klik tombol Next.
   catatan: install anaconda ke direktori/path yang tidak memiliki nama mengandung spasi atau karakter unicode.
   catatan: Jangan meng-install sebagai Administrator kecuali diinginkan hak akses admin di input setiap kali menjalankan Anaconda.



Install Anaconda di Windows

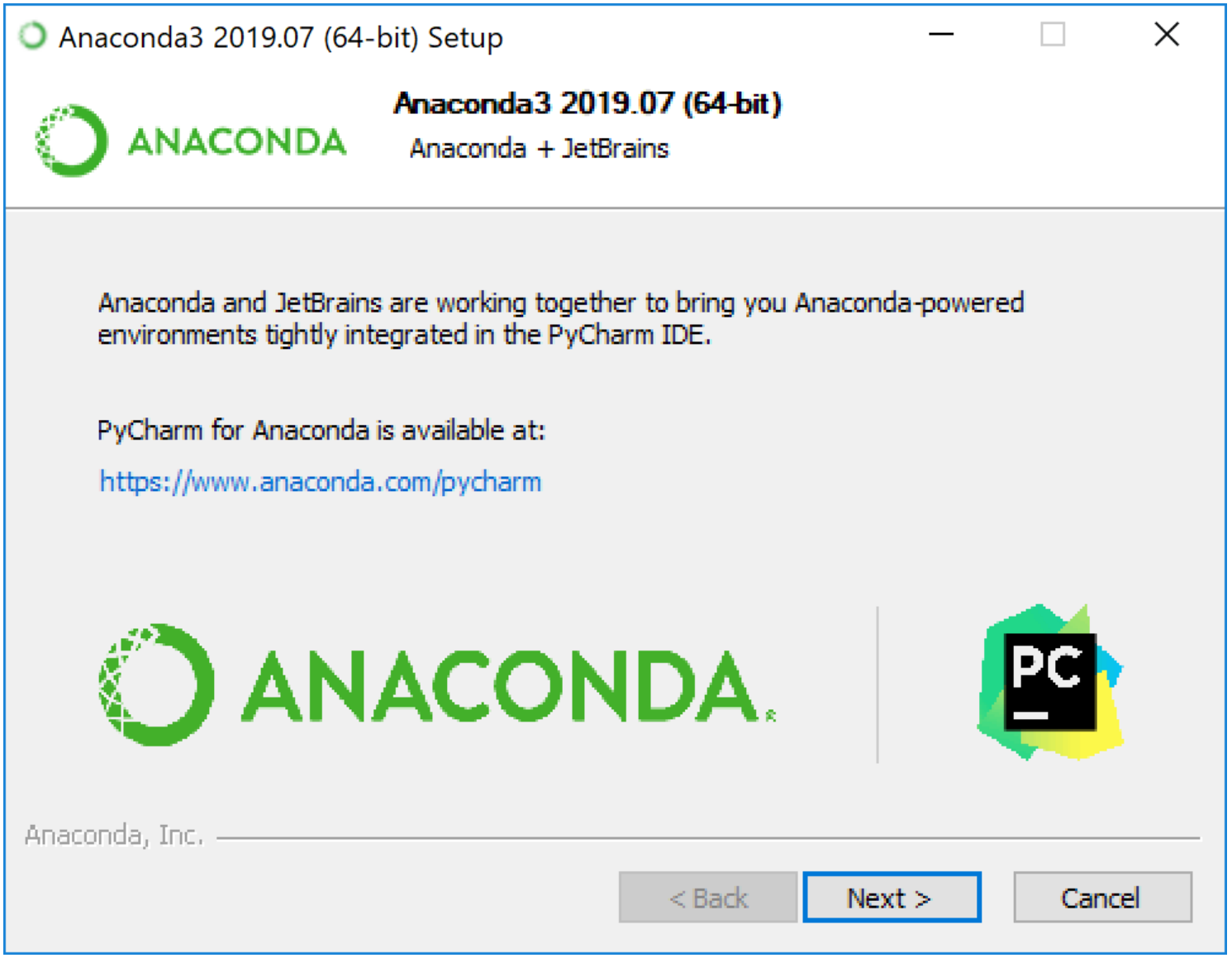
8. Pilih apakah untuk menambahkan PATH Anaconda ke environment variable windows anda. Disarankan tidak menambahkan PATH environment variable Anaconda, karena dapat mempengaruhi software lain. Cara gantinya, menggunakan sofware Anaconda dengan membuka Anaconda Navigator atau Anaconda Prompt dari Start Menu.



Install Anaconda di Windows

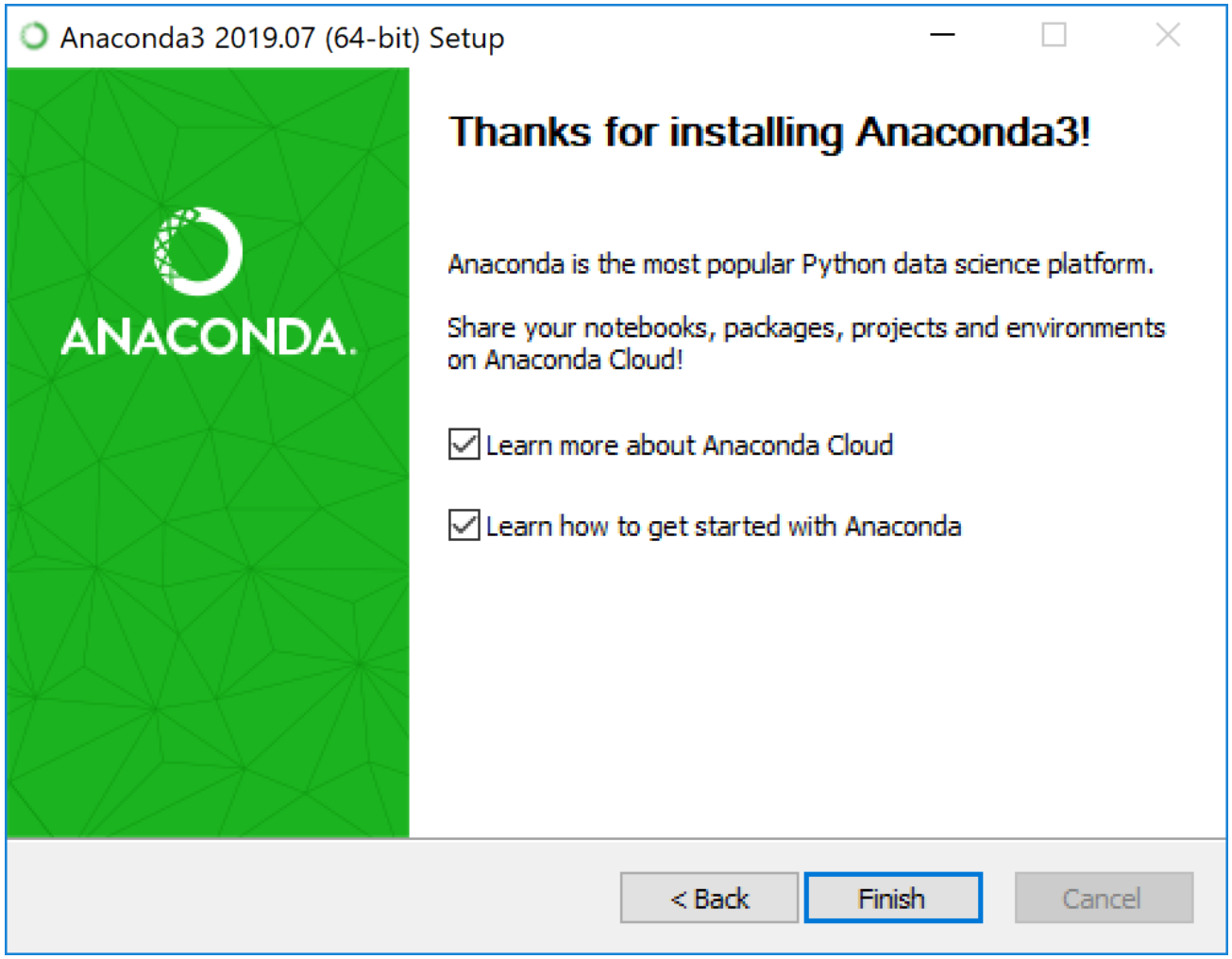
9. Pilih apakah untuk meregister Anaconda sebagai default Python anda. Kecuali anda berencana untuk meng-install dan menjalankan lebih dari satu versi Anaconda atau lebih dari satu versi Python. Terima default dan biarkan box nya tercentang.
10. Klik tombol Install. Jika anda ingin melihat paket[2] Anaconda yang sedang di install, klik Show Details.
11. Klik tombol Next.
12. Pilihan. Untuk meng-install PyCharm dengan Anaconda, klik link nya. Atau untuk meng-install Anaconda tanpa PyCharm, klik tombol Next.

Install Anaconda di Windows



Install Anaconda di Windows

13. Setelah sukse meng-install anda akan melihat dialog box “Thanks for installing Anaconda”.

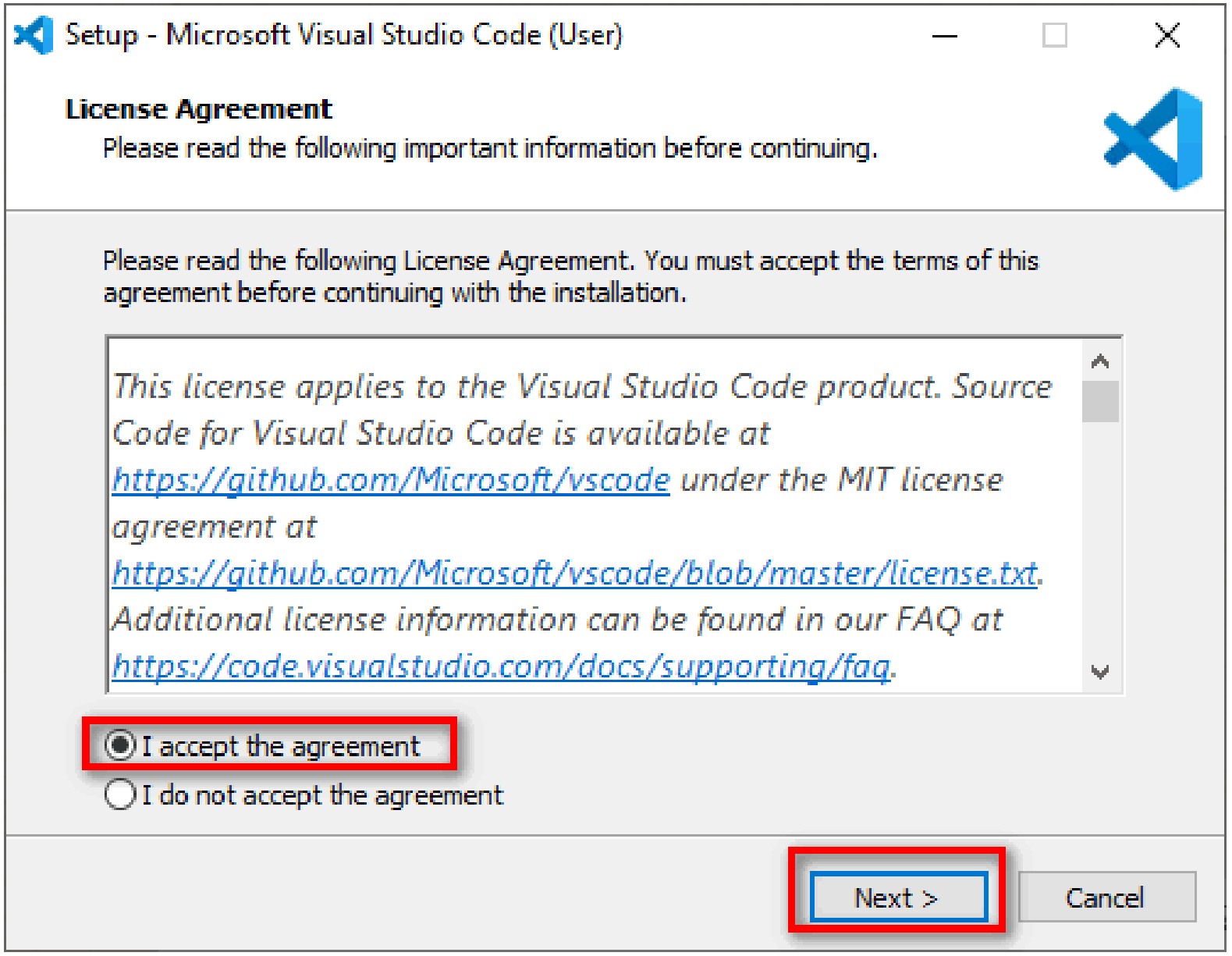


Install Visual Studio Code di Windows

Setelah berhasil di download VSCode, lanjut ke proses installasi.

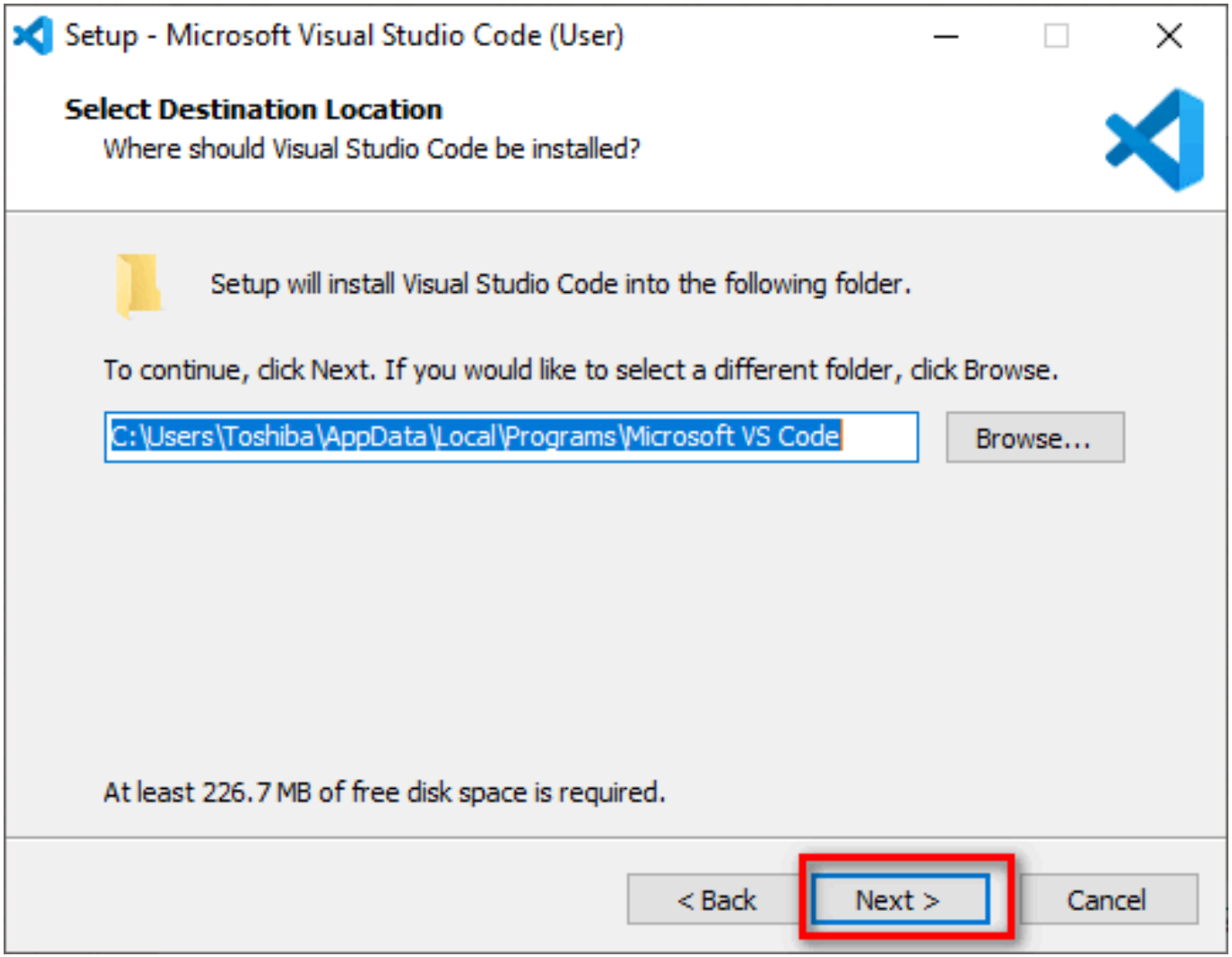
1. Double klik pada file installer nya atau klik kanan kemudian pilih Run as Adminstrator.
2. Jika muncul peringatan Run as Administrator, silahkan klik Yes.
3. Pilih “I accept the aggrement” untuk menyetujui “License Agreement”, kemudian klik Next.

Install Visual Studio Code di Windows



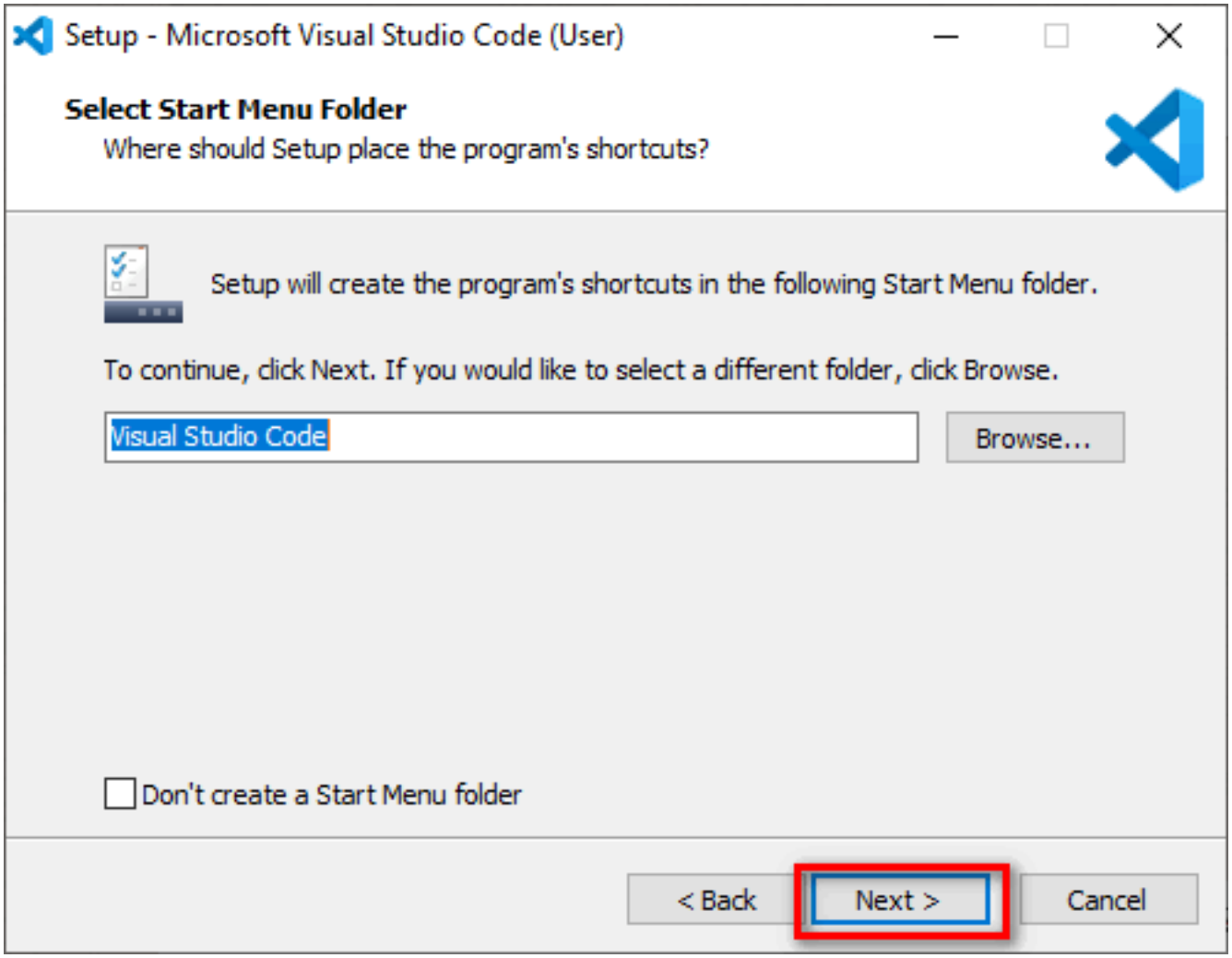
Install Visual Studio Code di Windows

4. Untuk Select Destination Location bisa di biarkan saja jika lokasi installasi tidak akan di rubah. Klik Next.



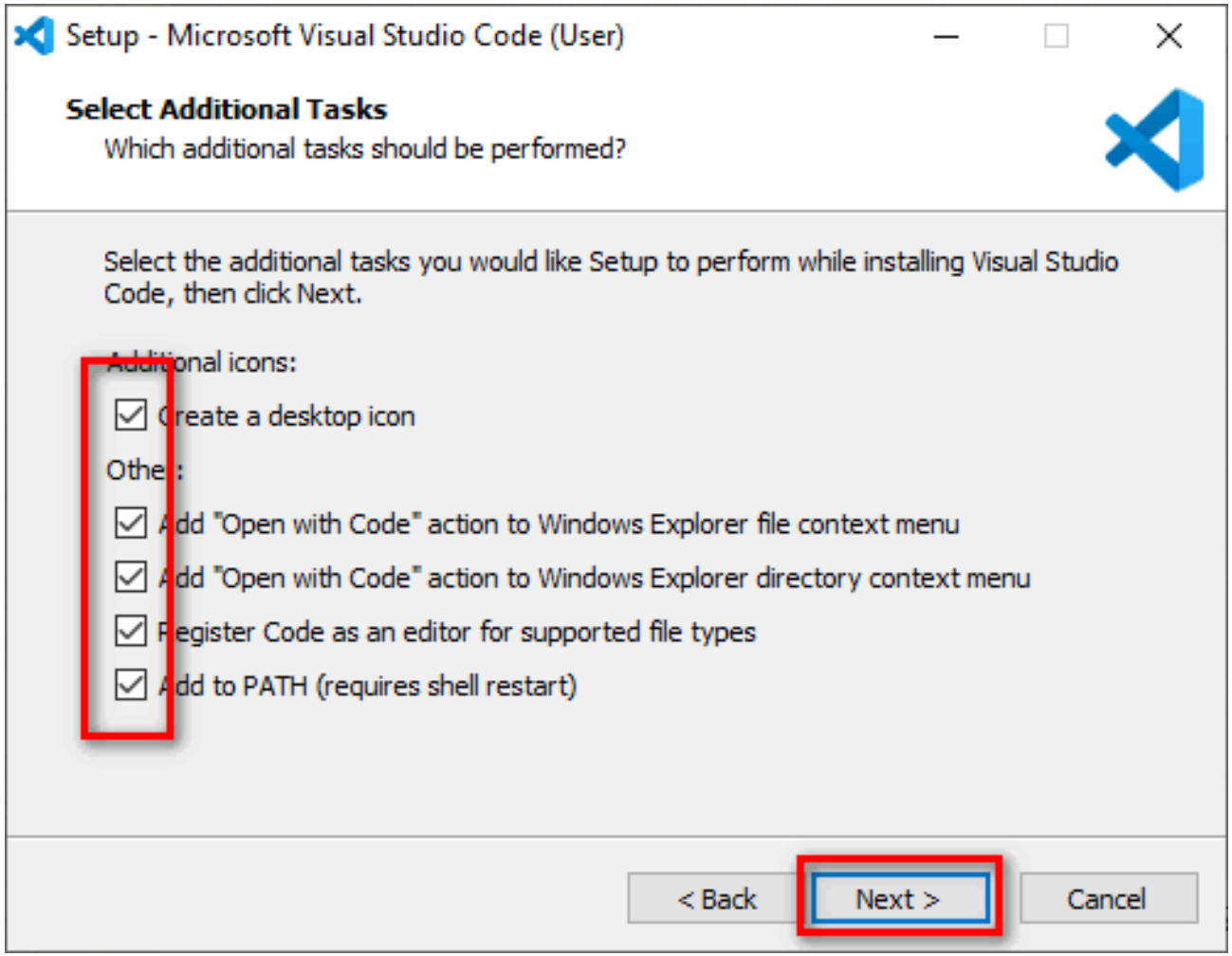
Install Visual Studio Code di Windows

5. Klik Next lagi jika tidak akan merubah Start Menu Folder.



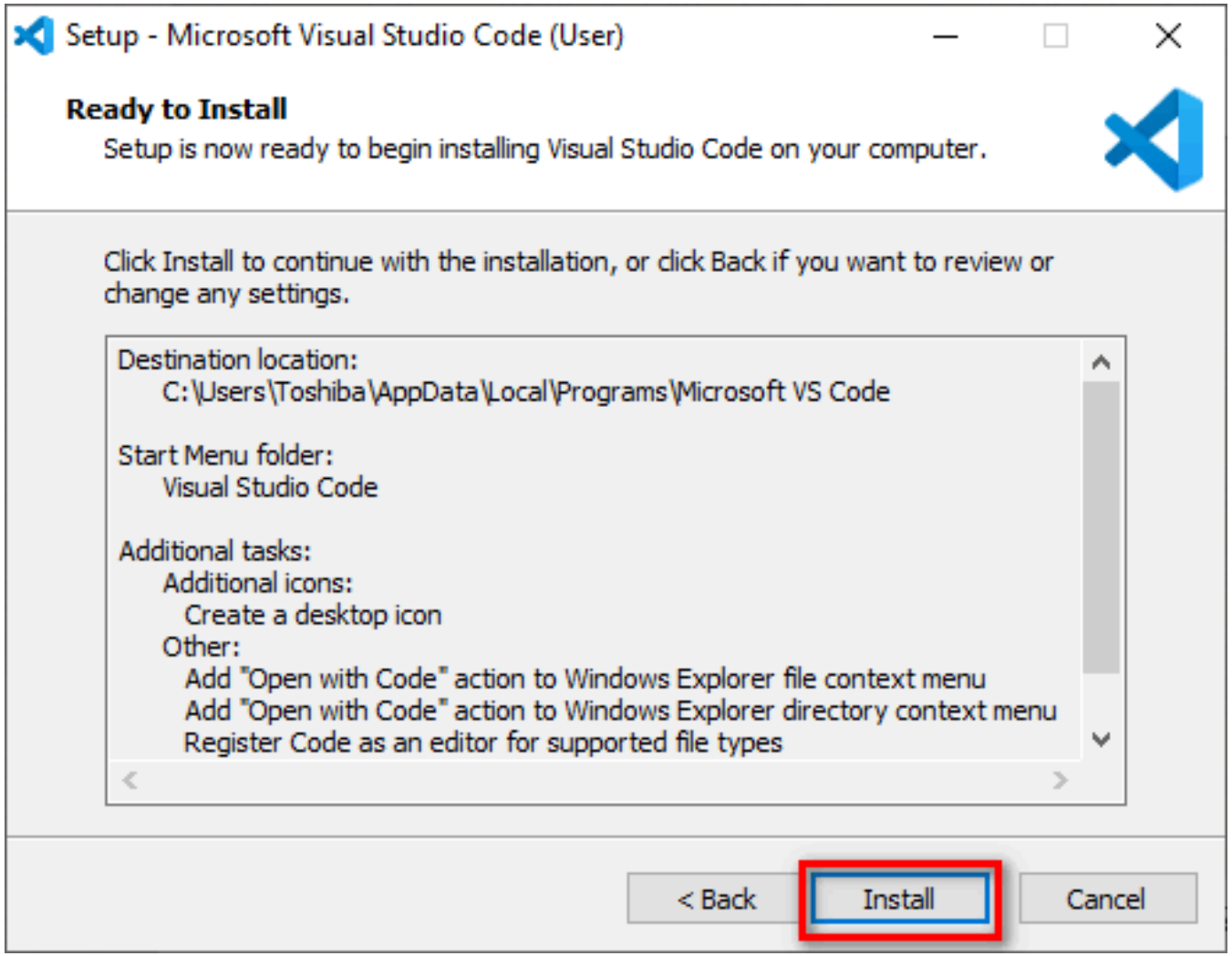
Install Visual Studio Code di Windows

6. Di bagian Select Additional Tasks centang semua. Kemudian Next.



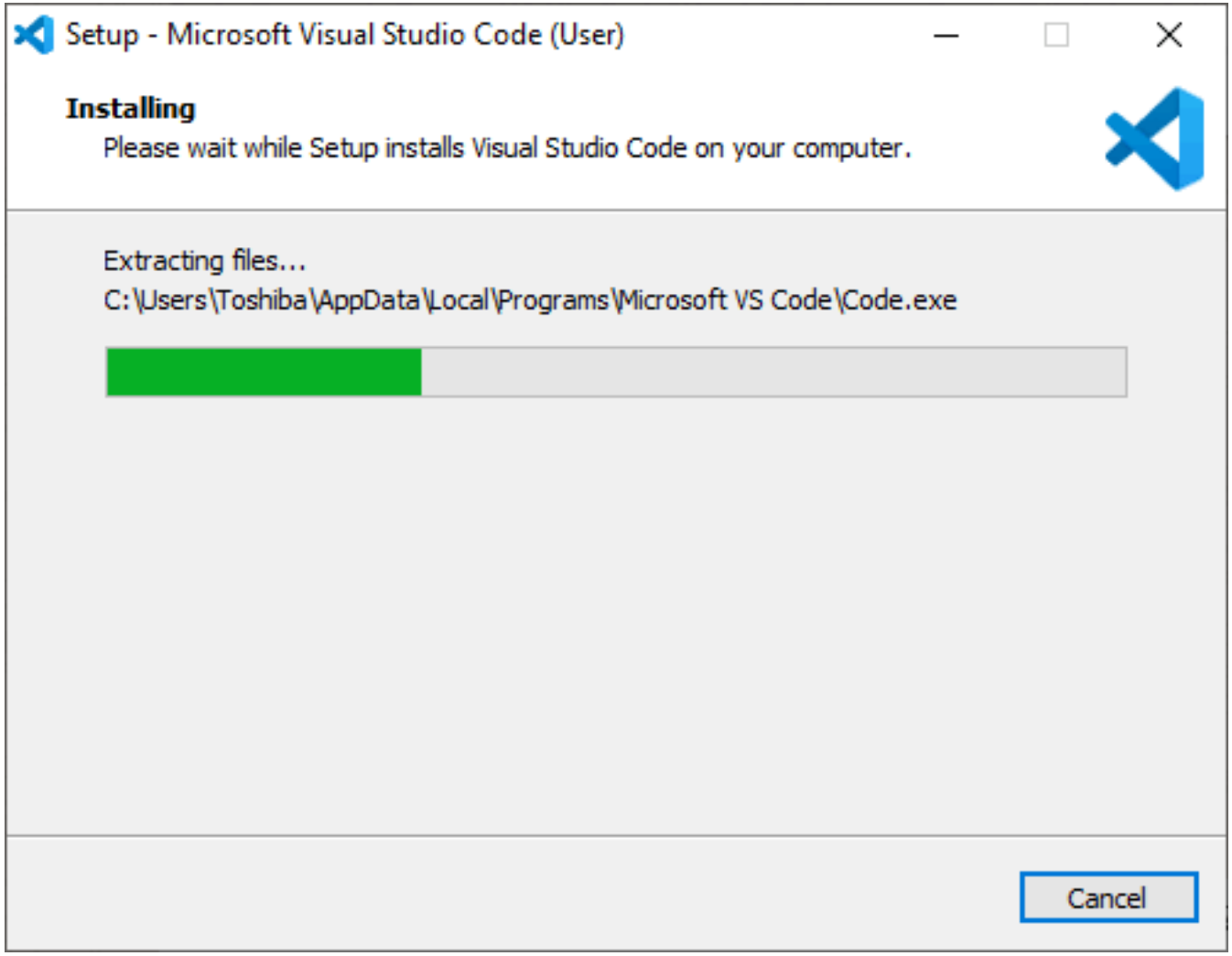
Install Visual Studio Code di Windows

7. Lalu klik Install untuk memulai proses installasi.



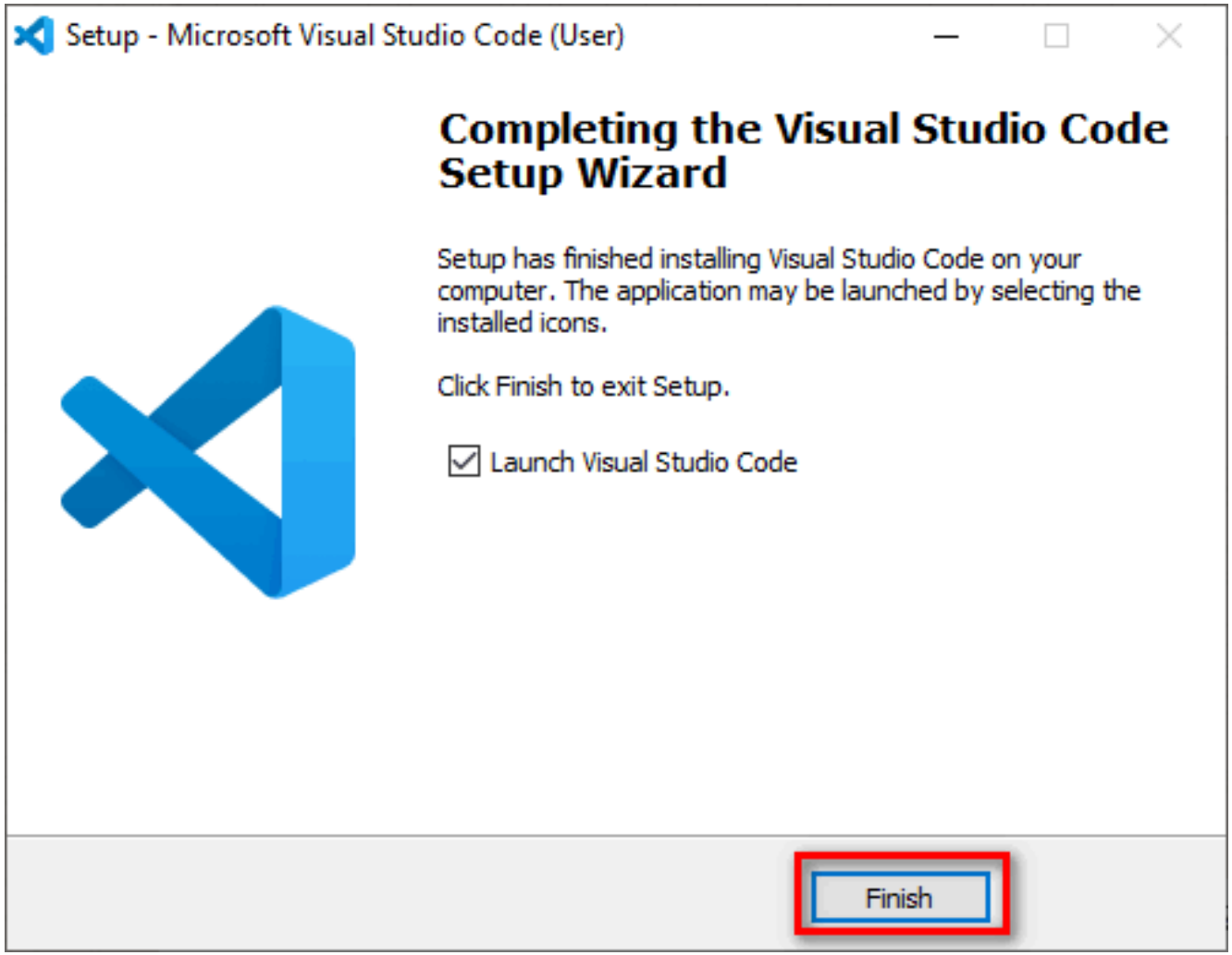
Install Visual Studio Code di Windows

8. Tunggu sampai proses installasi selesai.

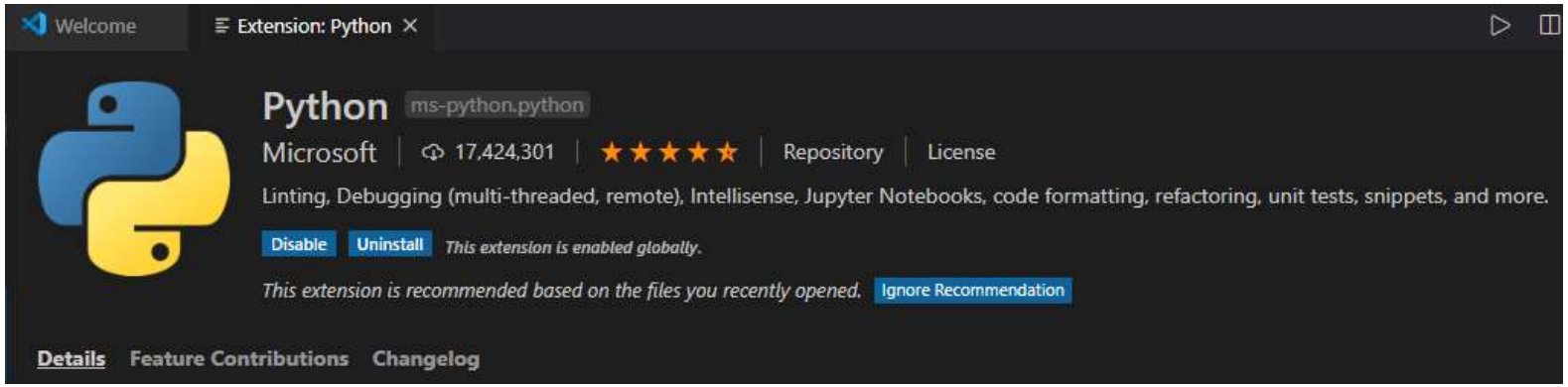


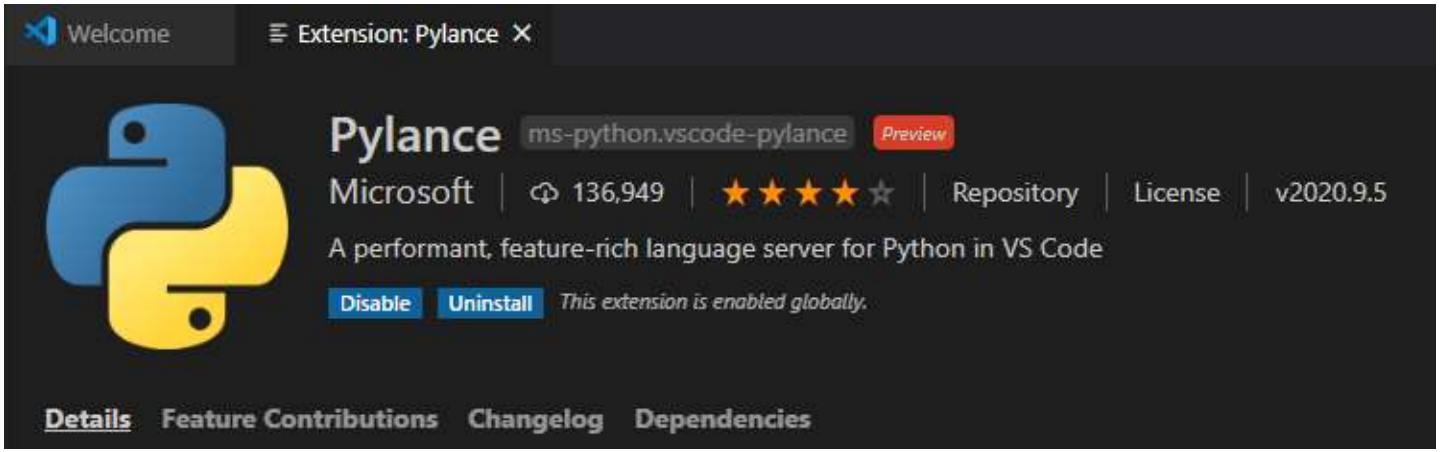
Install Visual Studio Code di Windows

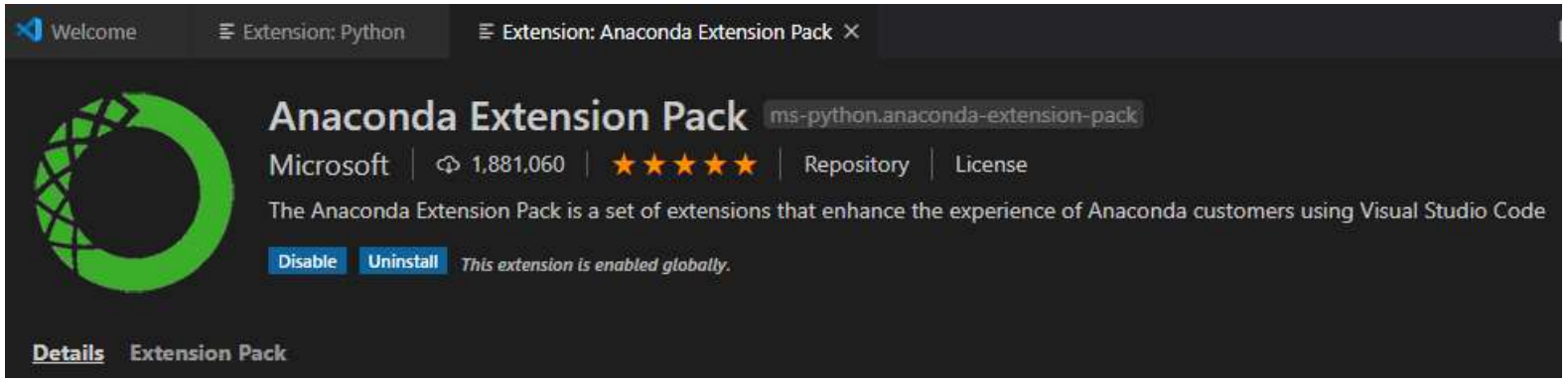
9. Setelah selesai klik Finish.



Install Extension Python di Visual Studio Code







Install Extension Python di Visual Studio Code

- Klik tab Extensions yang ada di sebelah kiri (1).
- Kemudian ketik “Python” di pencarian (2).
- Pilih Python (3).
- Kemudian klik Install pada Extensions: Python (4)

Why using Package Manager

Didalam pemrograman Python terdapat dua **Package Manager** yang terkenal yaitu **PIP** dan **Anaconda**. Alasan gunakan PIP/Anaconda :

Domino Effect Dependency Library problem:

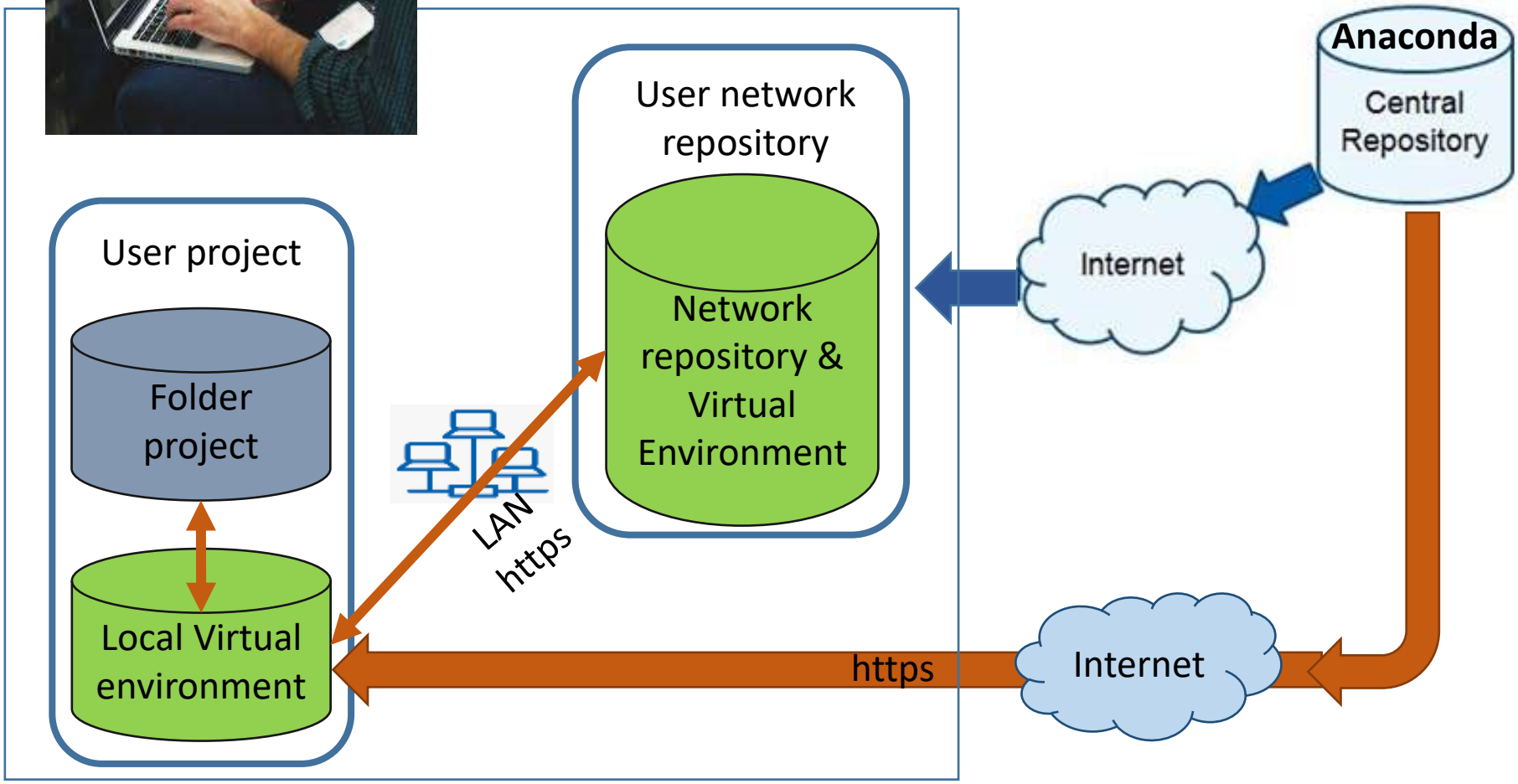
- Anggap dalam sebuah project kita butuh *library module* tambahan (mis, utk algoritma kriptografi). Sekarang kita akan coba menambahkan *library module* ke dalam project tadi.
- Dengan cara manual kita bisa mendownload semua file *library module* yang kita butuhkan dari internet dan menaruhnya didalam path project atau virtual environment. **Tentu cara manual ini cukup melelahkan**.
- Tetapi kadang tidak semudah itu, bisa jadi *library module* itu membutuhkan *dependency library* yang lain dgn versi tertentu dan *dependency library* itu membutuhkan *dependency library* yg lain lagi dan seterusnya. Istilahnya **domino effect dependency**.
- Bayangkan kalo harus mencari satu persatu, tentu sangat melelahkan. Inilah yang dinamakan dengan ***package library hell*** **(neraka paket librari)**.

Why using Package Manager

Searching Dependency Library problem:

- Dengan cara manual kita bisa mendownload semua file *library module* yang kita butuhkan dari internet tetapi kemungkinan besar bisa dari website yang berbeda-beda. **Tentu cara ini cukup melelahkan, dan ini disebut *searching library problem***.
- PIP dan Anaconda menyediakan ***online repository*** yang memudahkan kita untuk download secara otomatis library yang kita butuhkan + dependency nya. Sehingga hanya perlu satu website untuk mendapatkan *libray module* yang dibutuhkan.
- Dalam hal ini Anaconda lebih unggul dari PIP karena Anaconda memiliki komputasi **yang lebih cerdas** untuk memverifikasi library modul dan versi nya terhadap *dependency* antar *library module* tersebut sebelum mendownload-nya ke project (atau venv) user.

Why using Package Manager



Why using Package Manager

- Secara umum PIP dan Anaconda akan membantu kita (programmer) untuk mencari dan men-*download library module* yang diperlukan pada ***online repository-nya,*** sekaligus menganalisis paket-paket (dan versi) *library module* yang sudah ada pada **virtual environment** programmer untuk dapat memberikan versi *library module* yang diperlukan dimana yang sesuai dengan lingkungan **venv** nya.
- PIP dan Anaconda juga digunakan untuk mengelola **virtual environment (venv)** Python pada lingkungan lokal atau network programmer. Mengenai konsep **venv** Python akan dijelaskan pada bab berikutnya.

Tugas

Anda diminta untuk menginstall Anaconda dan IDE VScode.

Jawaban tugas ini adalah :

1. Screenshoot capture aplikasi Anaconda Navigator dan VScode yang dijalankan dari icon aplikasinya.

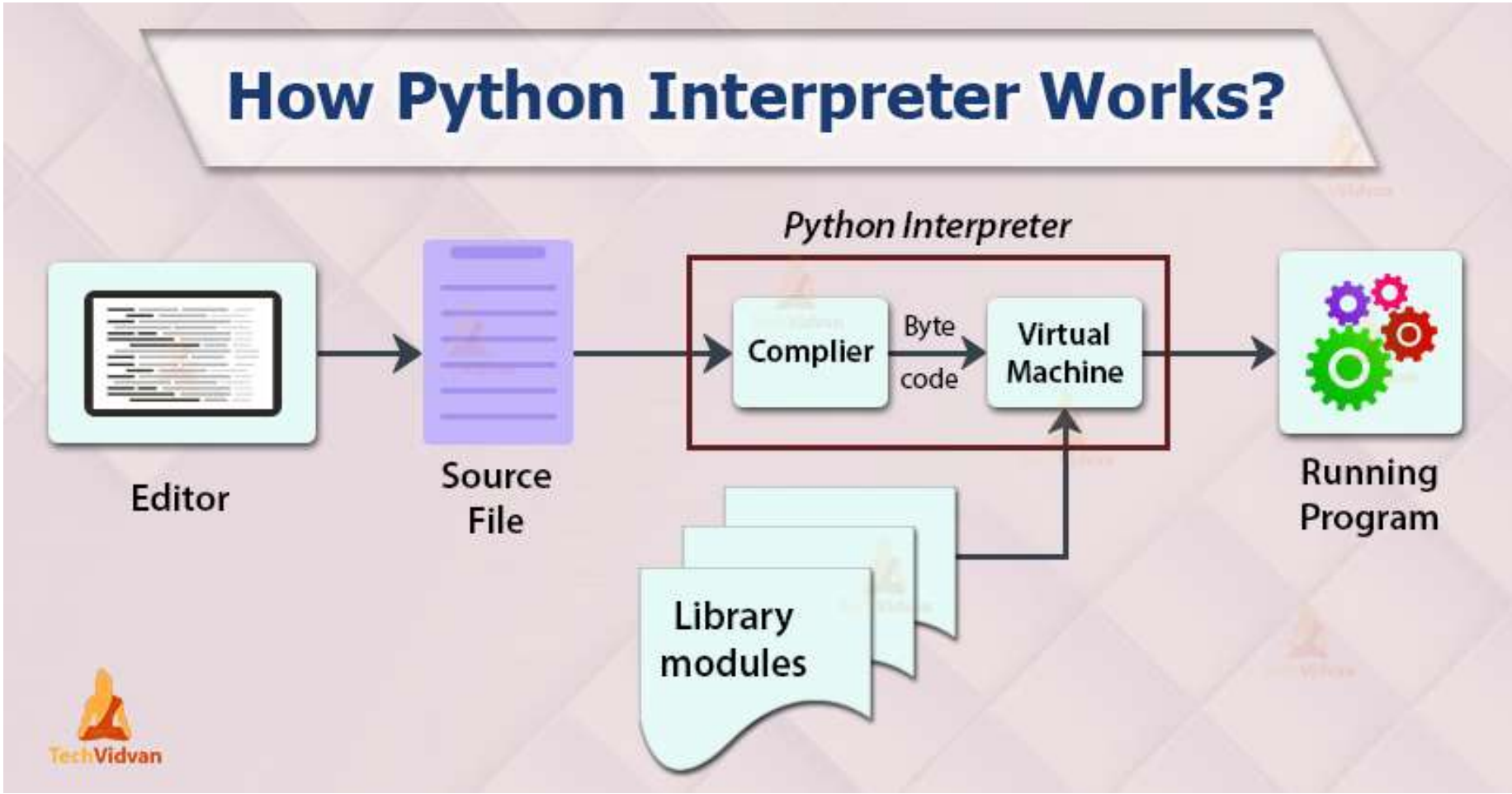
Hasil screenshoot bisa di rubah menjadi format pdf atau apapun sehingga sedemikian resolusi atau ukuran file total tidak melebihi 2MB seperti yang dipersyaratkan ketika mengupload jawaban anda ke elearning.istn.ac.id.

Thank You

Programming for Science

Memulai Pemrograman Python

Bagaimana Python Interpreter bekerja ?



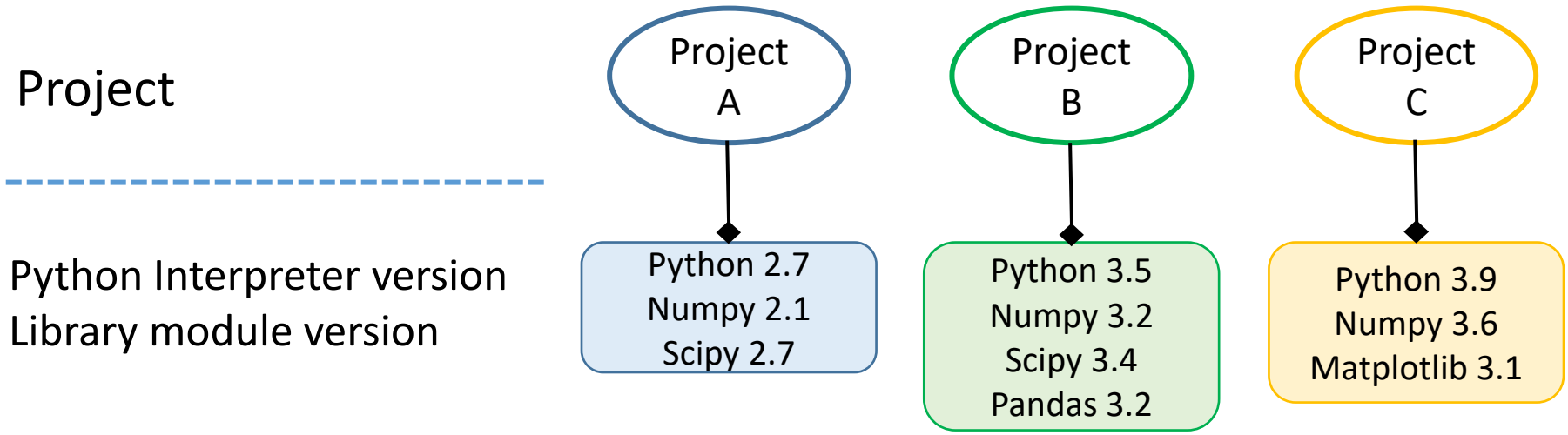
Python project.
Write the program code.

Compiling the program.
Execution environment.

Output program.

Konsep Virtual Environment Python

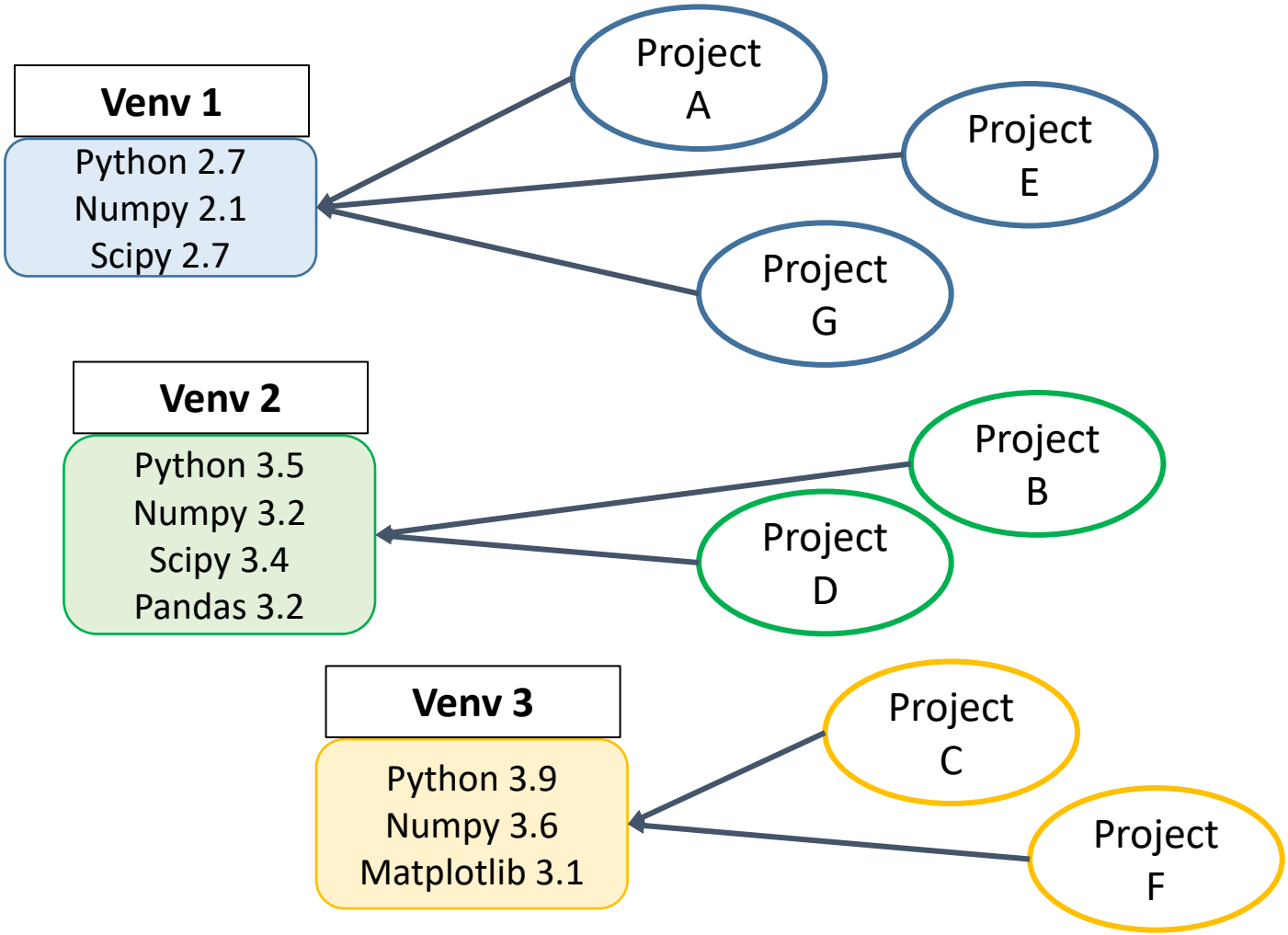
- Terkadang python project yang berbeda memerlukan versi python dan library modules yang berbeda agar project yang dibangun dapat *running* dengan baik.



- Versi Python Interpreter ataupun Library Module yang sama hanya bisa memiliki satu versi dalam *execution environment* yang sama.
- Disinilah dibutuhkan ***Virtual Execution Environment*** (disingkat *virtual environment* = venv). Sebagai tempat yang terisolasi untuk versi python dan library modules yang berbeda sesuai kebutuhan project-project yang ada.

Konsep Virtual Environment Python

- Virtual Environment (venv) bisa digunakan (berbagi) oleh project-project yang ada.

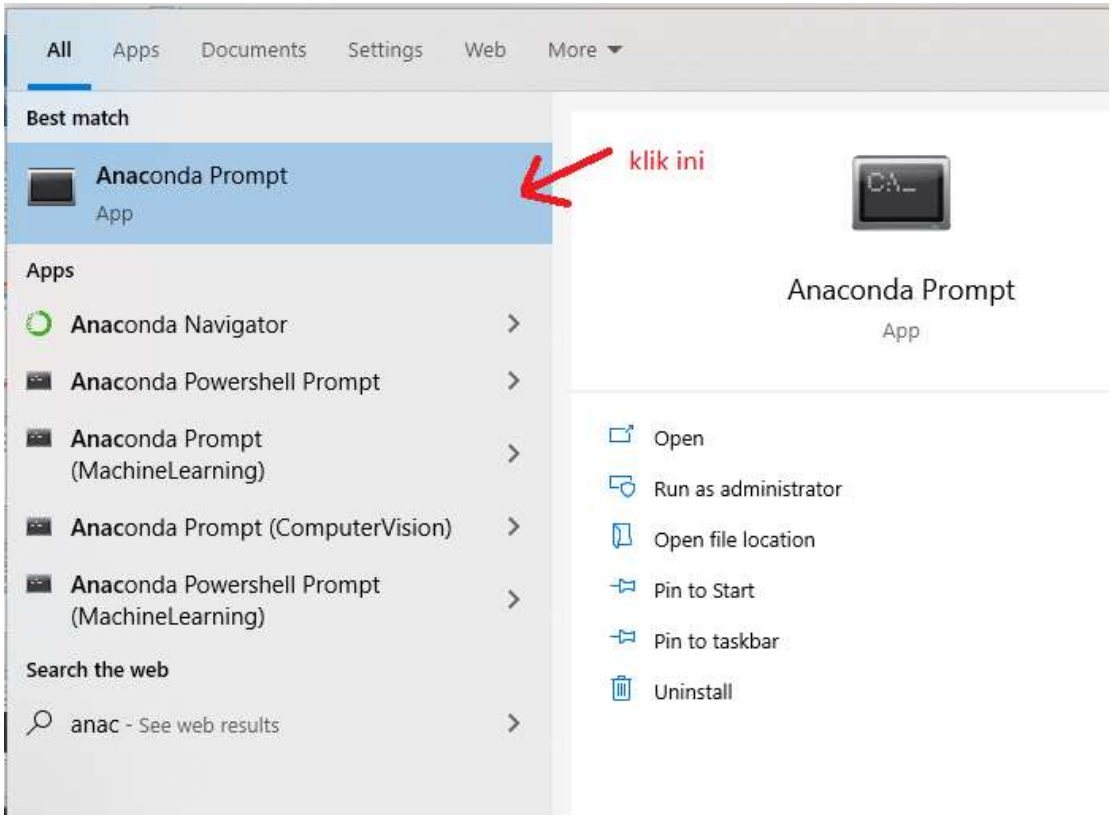


Menggunakan Python

- Buat Virtual Environment menggunakan Anaconda prompt. Sebagai python environment khusus yang terisolasi.
- Python environment baru ini digunakan untuk project^2 python kita, yang tidak mengganggu python environment default (base) bawaan Anaconda.
- Buat folder kerja, tempat menaruh project python kita.
- Pindahkan python environment pada project yang dibuat ke python environment yang baru dibentuk.

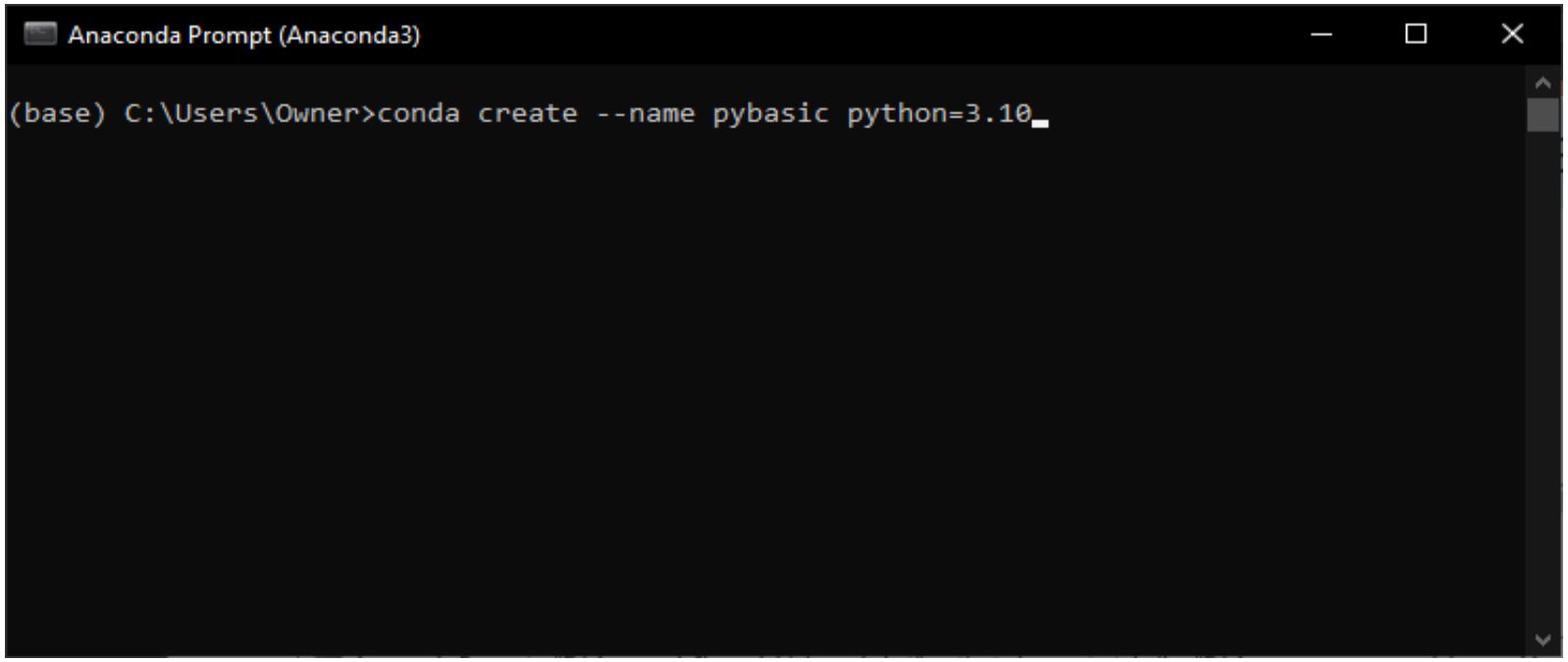
Membuat Virtual Environment Anaconda

- Buka Anaconda command prompt.



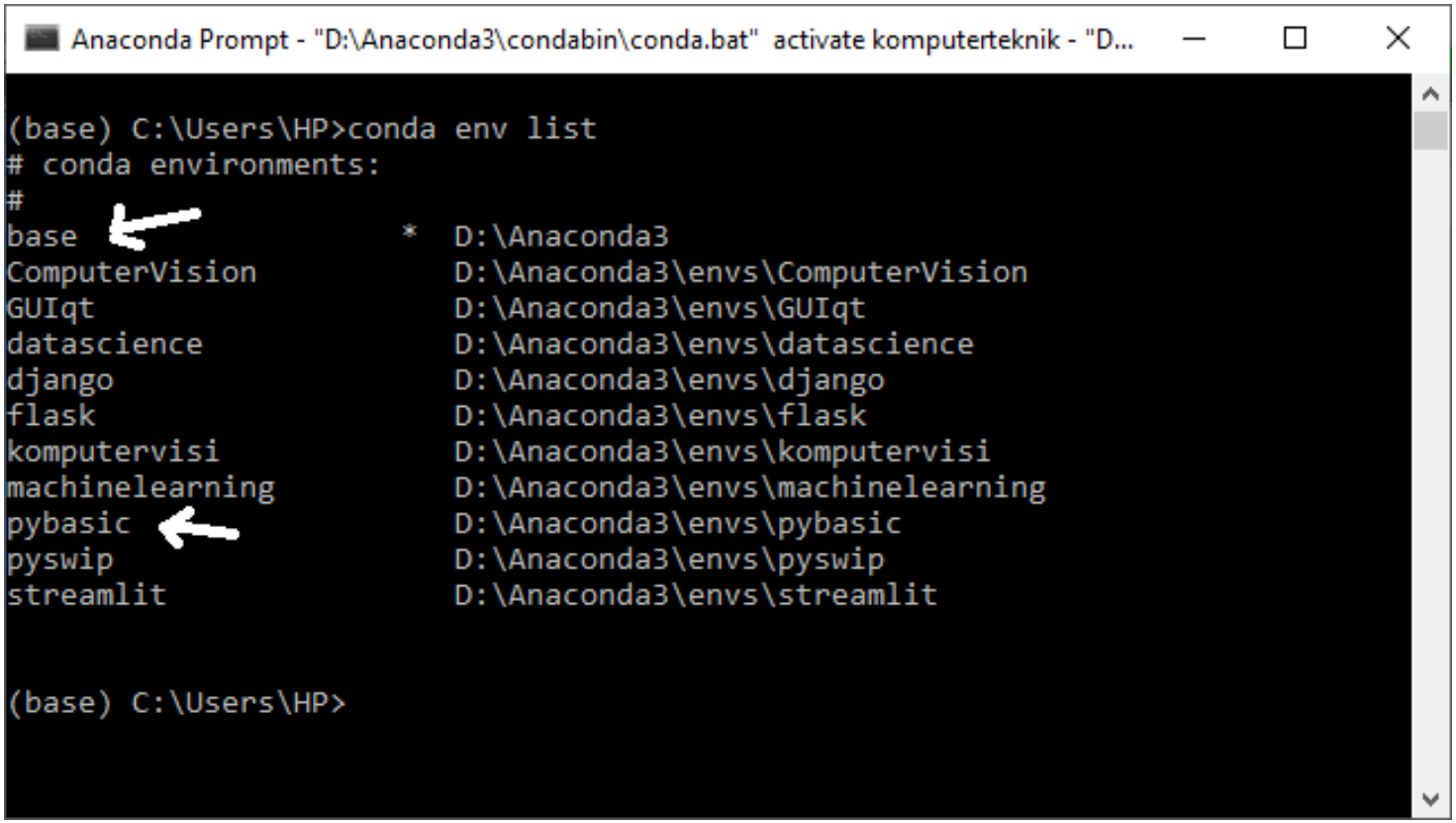
Membuat Virtual Environment Anaconda

- Pastikan komputer terhubung ke internet.
- Kita akan membuat virtual environment python dengan nama **pybasic** dan berisi **python versi 3.10**. Anda boleh menggunkan versi python yang lebih baru.
- Buka Anaconda command prompt. Untuk membuat venv ‘pybasic’, ketik seperti dibawah dan enter. Tunggu hingga selesai.



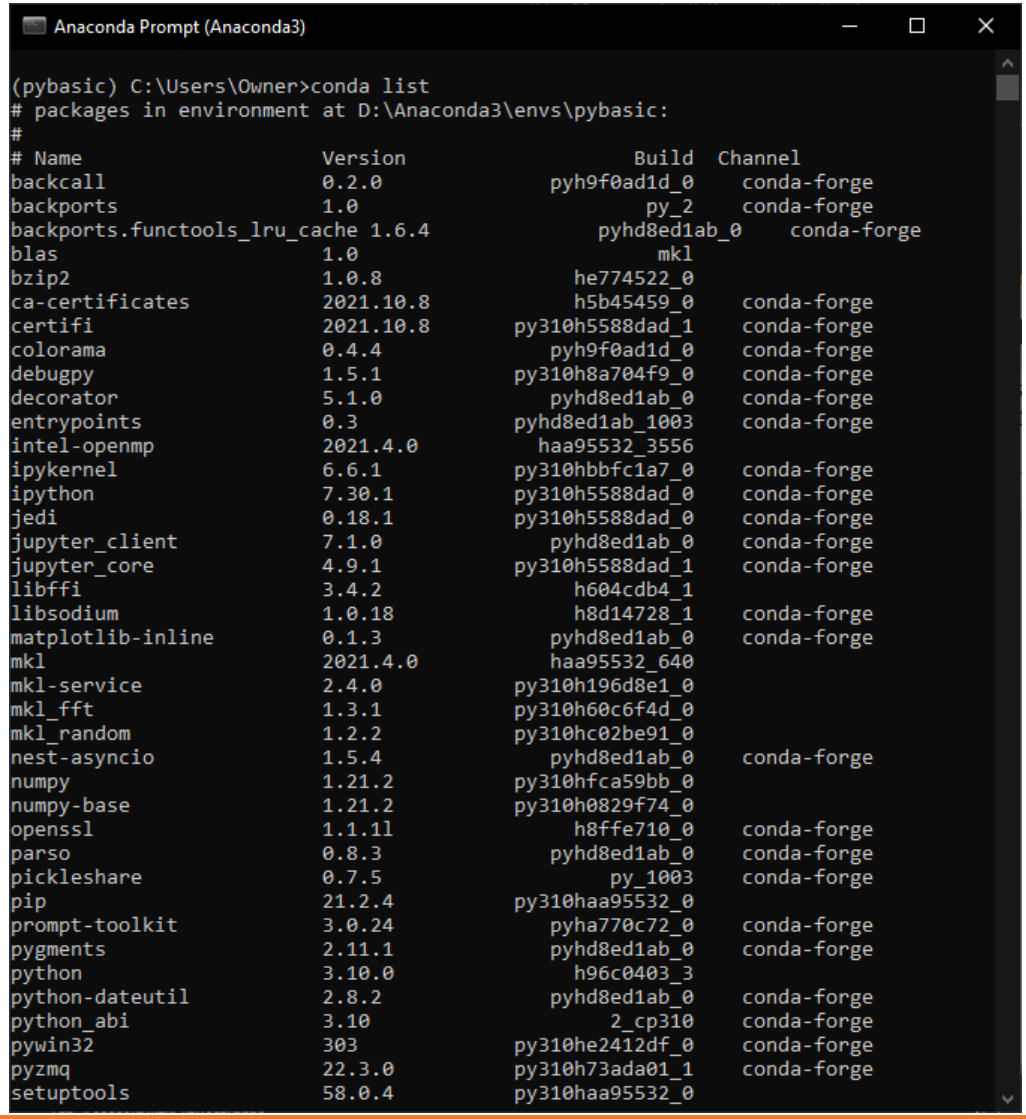
Membuat Virtual Environment Anaconda

- Untuk melihat hasil venv yg kita buat tadi maka pada Anaconda command prompt, ketik seperti dibawah dan enter. Pastikan Virtual Environment yang terlist adalah base dan pybasic (panah putih).



Membuat Virtual Environment Anaconda

- Untuk melihat isi *library modules* pada venv ‘pybasic’ maka pada Anaconda command prompt, ketik seperti dibawah dan enter. Pastikan tampilan spt dibawah.



Membuat Virtual Environment Anaconda

- Sampai tahap ini jika tampilan seperti di atas maka anda telah berhasil membuat Virtual Environment baru anaconda bernama “pybasic” yang akan kita pakai selama perkuliahan ini.
- Tutup saja window anaconda prompt.
- Kita lanjut ke aplikasi visual studio code sebagai editor untuk pengetikkan kode program dan sekaligus running program untuk melihat hasilnya.

Memulai Python Dengan VScode

- Konsep ‘Project’ pada pembuatan aplikasi akan diimplementasi menjadi sebuah folder (direktori) dimana isi dari folder tersebut adalah berbagai file program dan file lain yang diperlukan pada project tersebut.
- Buka command prompt.
- Buat folder kerja, tempat menaruh project python kita.
- Buka visual studio code.

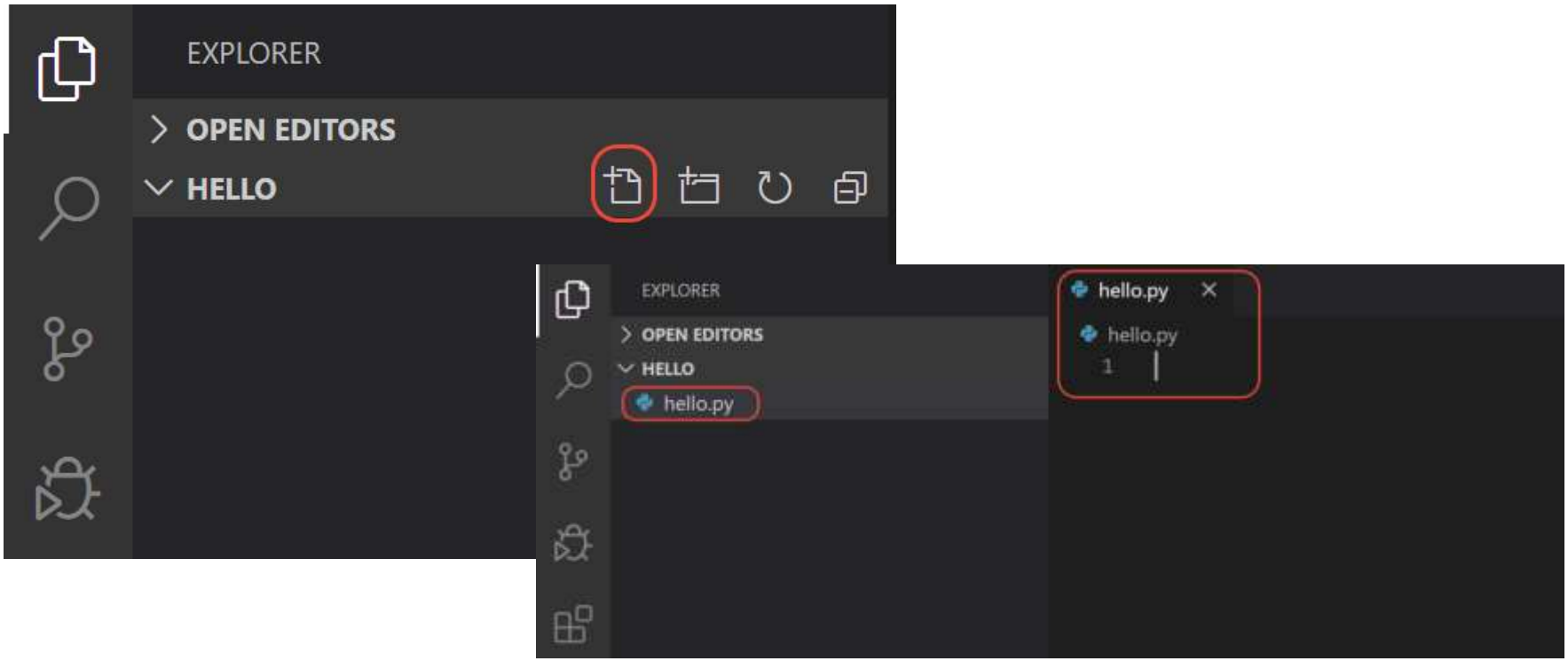
```
mkdir hello
cd hello
code .
```

Membuka VScode juga bisa dilakukan melalui icon start menu pada Sistem Operasi (Windows).

Selanjutnya didalam VScode anda memilih folder project yang diinginkan untuk dibuka.

Memulai Python

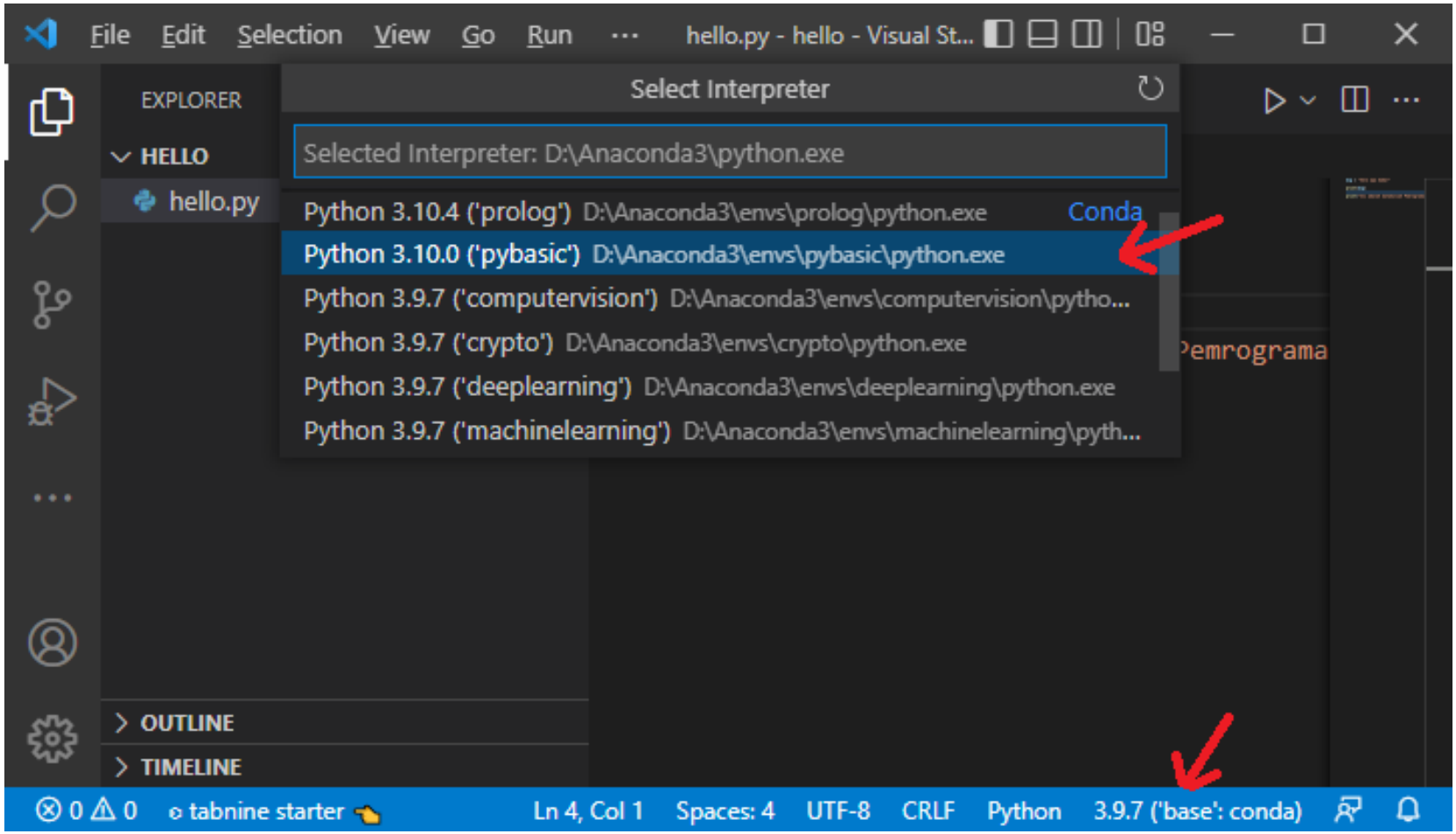
- Dari toolbar File Explorer, pilih button **New File** pada folder target.
- Namai file tersebut dengan hello.py dan otomatis akan terbuka editor pengetikan program di sebelah kanan.

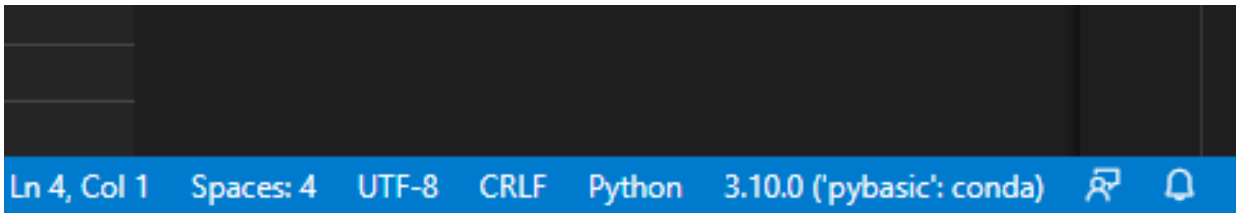


Memilih Interpreter Python

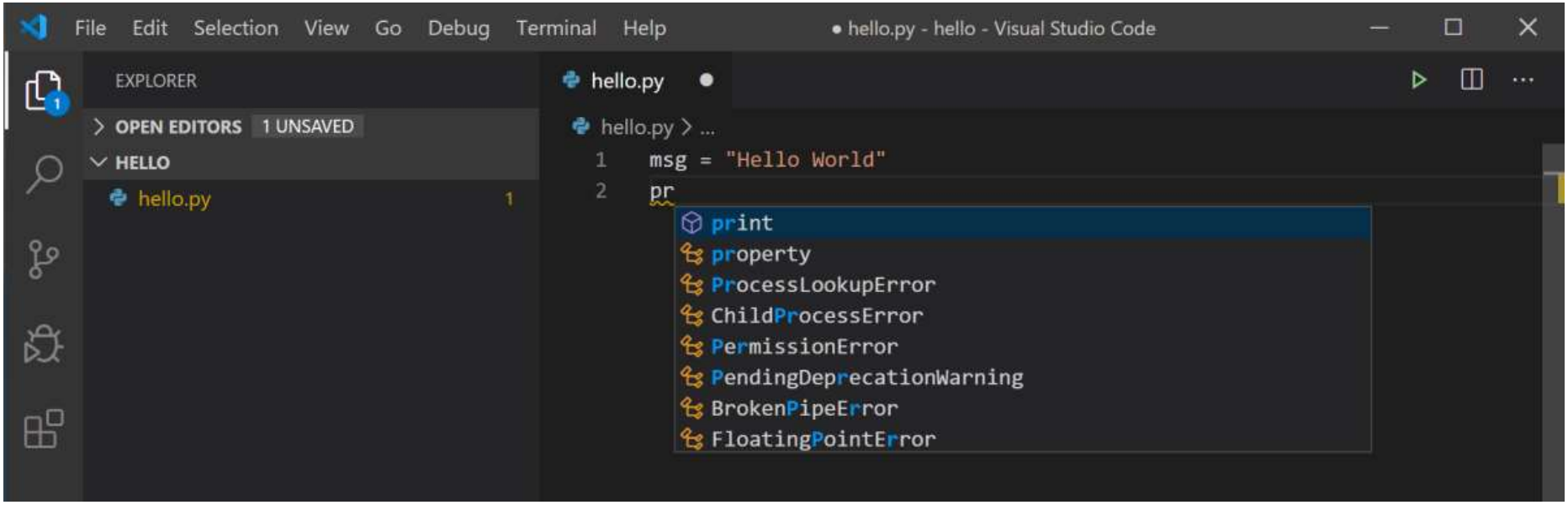
- Python adalah bahasa pemrograman interpreter, agar kode python dapat di eksekusi dan mendapatkan Python IntelliSense, kita harus memberitahu VS Code interpreter virtual environment mana yang digunakan.
- Dari dalam VS Code buka **CommandPalette** (Ctrl-Shift+P), mulai ketik **Python: Select Interpreter**.
- Kita juga bisa memilih python environment dengan memilih opsi **Select Python Environment** pada bagian kanan bawah window Visual Studio. Arahkan virtual environment ke “pybasic”.

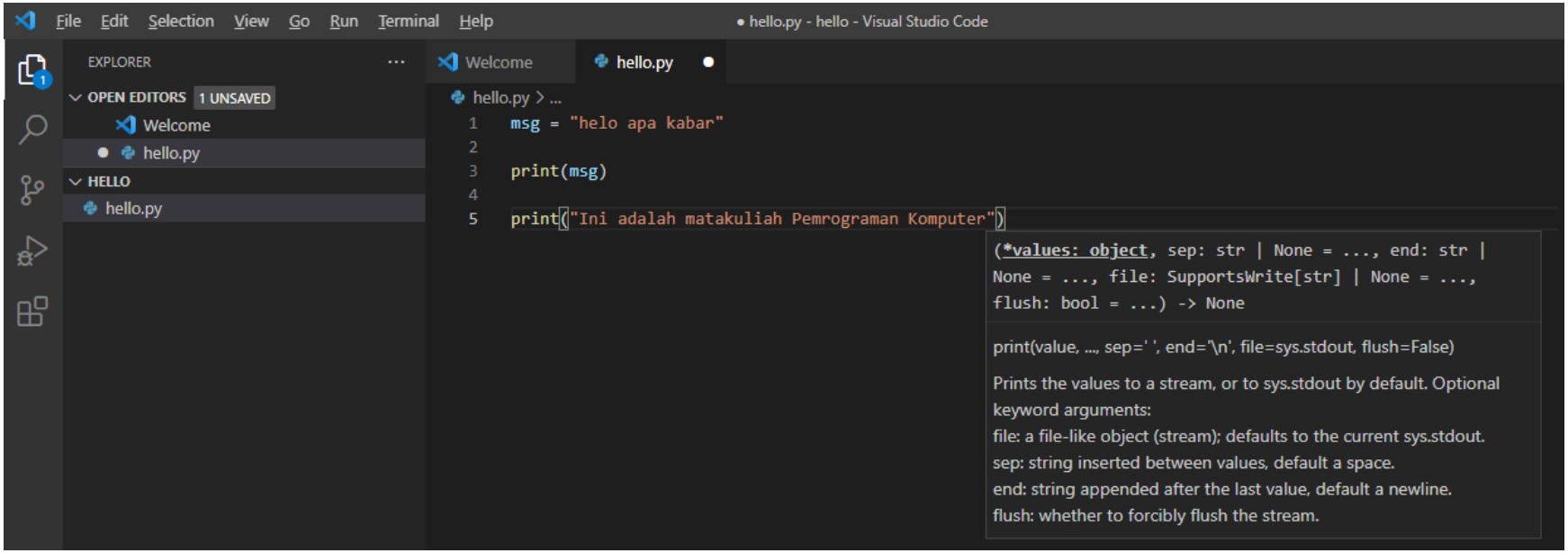
Memilih Interpreter Python





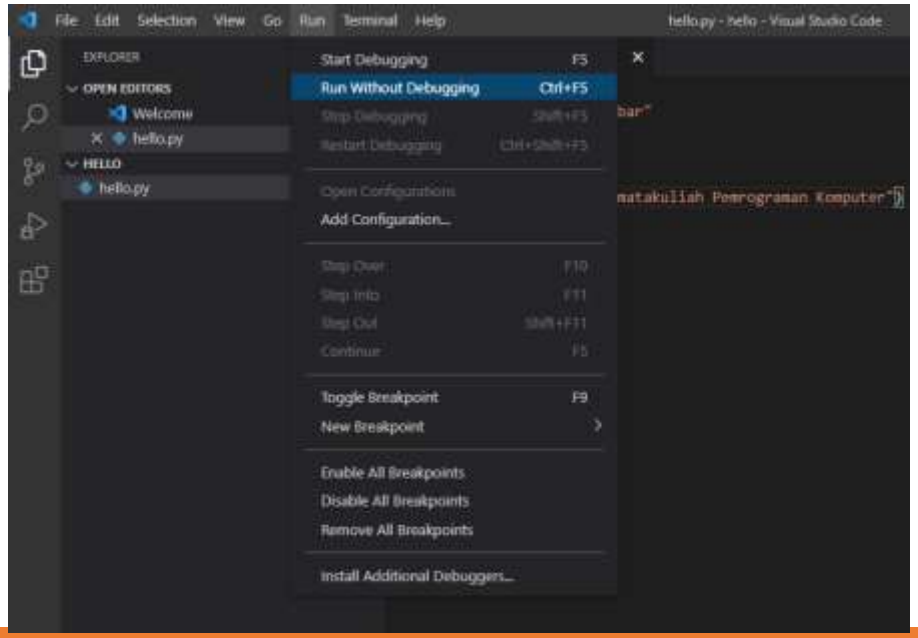
Memulai Python



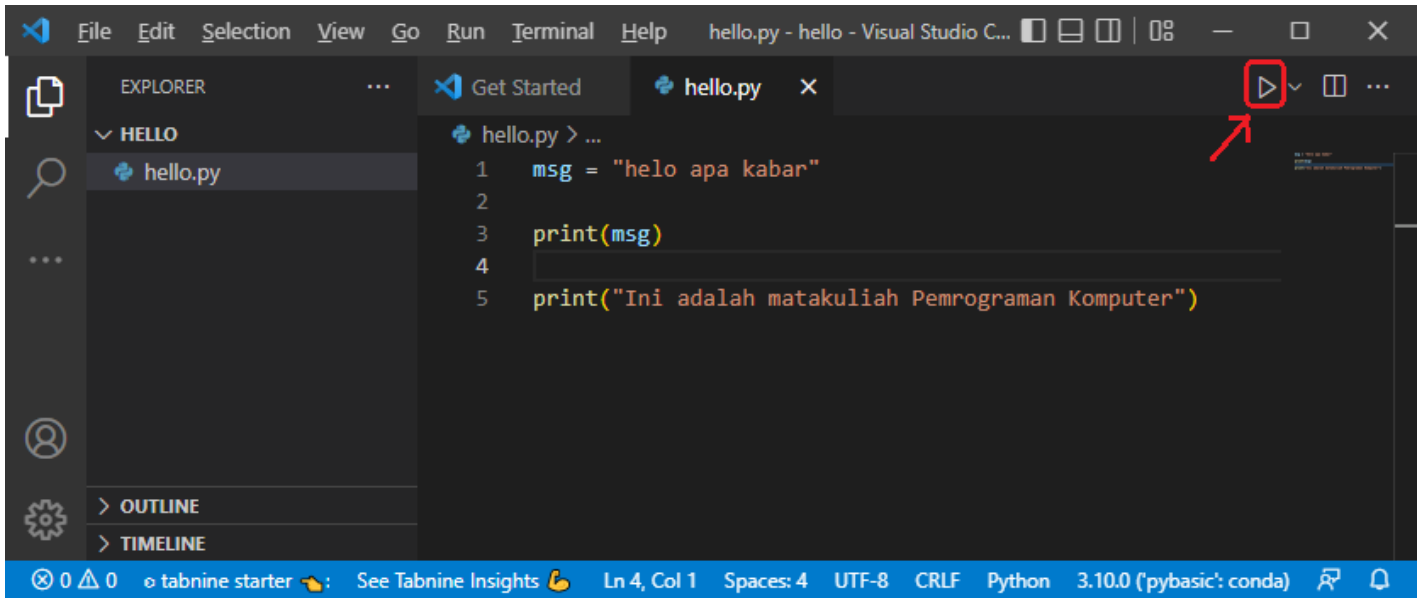


Memilih Interpreter Python

- Save atau mengetik ‘Ctrl-s’ untuk menyimpan file kode python ‘hello.py’.
- Selanjutnya kita akan mencoba untuk menjalankan program ‘hello.py’ dan melihat hasilnya. Yaitu dengan mengklik icon tombol  atau dengan memilih menu Run → Run Without Debugging, atau dengan mengetik *shortcut* ‘Ctrl+F5’.



Running Program Python



Hasil output *running* program adalah :

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  JUPYTER  COMMENTS        Python

Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

Loading personal and system profiles took 1514ms.
(base) PS D:\PROJECTS-PYTHON\PythonBasics\hello> conda activate pybasic
(pybasic) PS D:\PROJECTS-PYTHON\PythonBasics\hello> & D:/Anaconda3/envs/pybasic/python.exe d:/PROJECTS-PYTHON/PythonBasics/hello/hello.py
helo apa kabar
Ini adalah matakuliah Pemrograman Komputer
(pybasic) PS D:\PROJECTS-PYTHON\PythonBasics\hello>
```

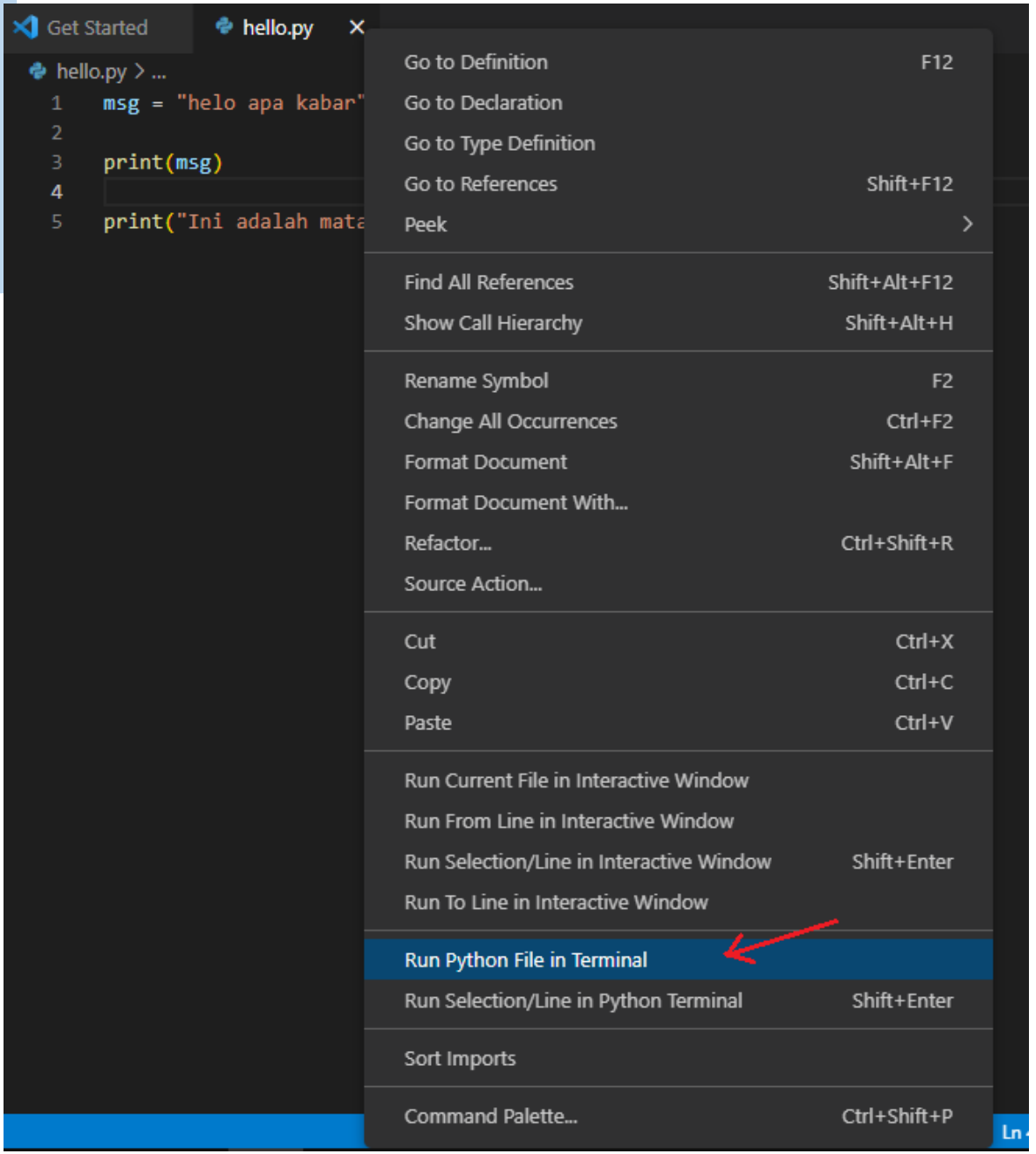
Running Program Python

Untuk menjalankan file kode program juga bisa dilakukan dengan cara :

a) Tempatkan kursor pada area pengetikkan program.

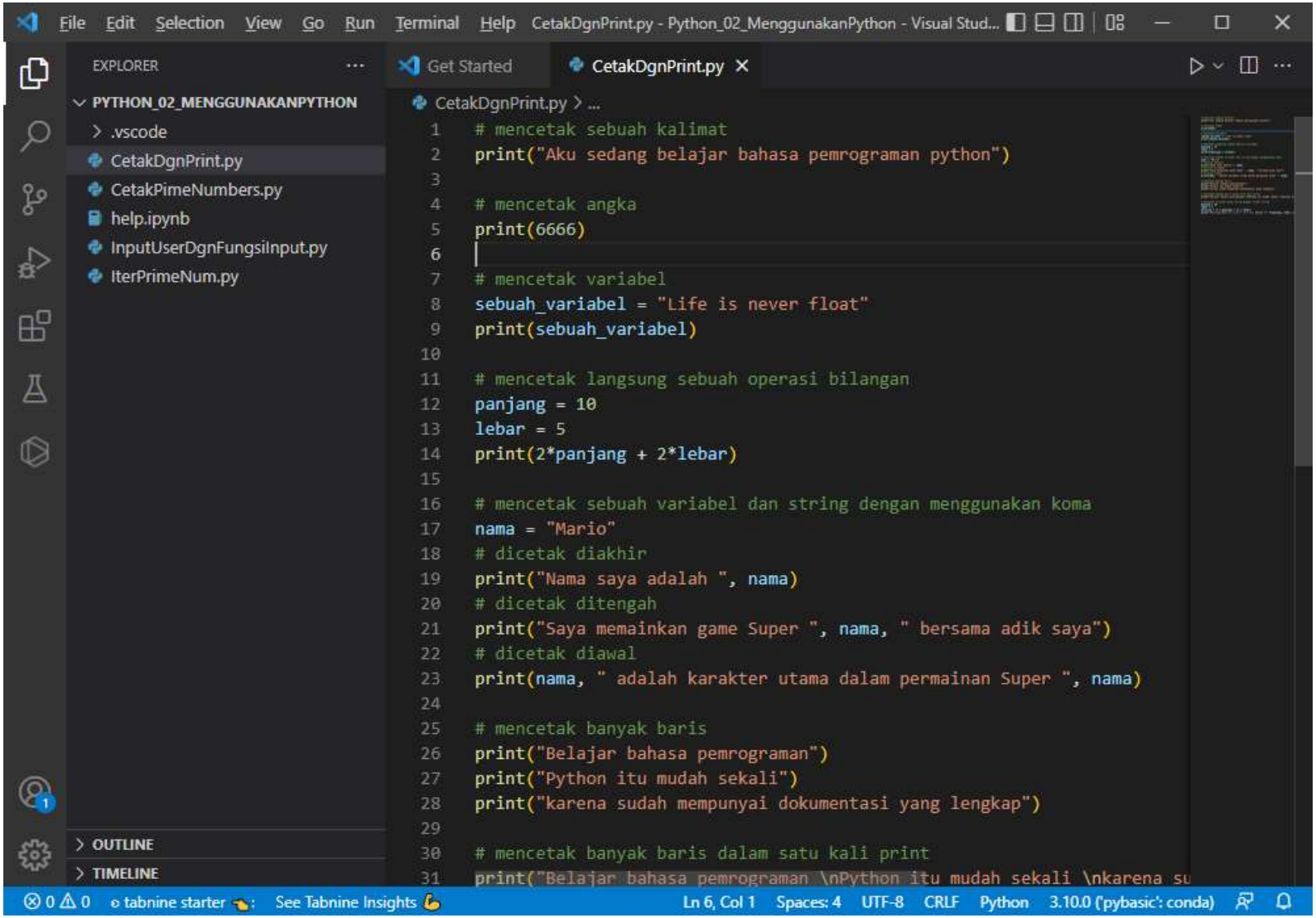
b) Kemudian klik button mouse kanan.

c) Pada menu yang tampil pilih menu ‘Run Python File in Terminal’.



Mencetak Informasi Dgn Print()

- Ketik program python seperti pada tampilan berikut :



Mencetak Informasi Dgn Print()

- Listing program lengkapnya ada pada file teks notepad yang disertakan bersama modul ini atau pada halaman berikutnya.
- Jalankan (running) program tersebut dan apa hasil output dari program ??

Mencetak Informasi Dgn Print()

Get Started | CetakDgnPrint.py

CetakDgnPrint.py > ...

```python
# mencetak sebuah kalimat
print("Aku sedang belajar bahasa pemrograman python")

# mencetak angka
print(6666)

# mencetak variabel
sebuah_variabel = "Life is never float"
print(sebuah_variabel)

# mencetak langsung sebuah operasi bilangan
panjang = 10
lebar = 5
print(2*panjang + 2*lebar)

# mencetak sebuah variabel dan string dengan menggunakan koma
nama = "Mario"
# dicetak diakhir
print("Nama saya adalah ", nama)
# dicetak ditengah
print("Saya memainkan game Super ", nama, " bersama adik saya")
# dicetak diawal
print(nama, " adalah karakter utama dalam permainan Super ", nama)

# mencetak banyak baris
print("Belajar bahasa pemrograman")
print("Python itu mudah sekali")
print("karena sudah mempunyai dokumentasi yang lengkap")

# mencetak banyak baris dalam satu kali print
print("Belajar bahasa pemrograman \nPython itu mudah sekali \nkarena sudah mempunya dokumentasi yang lengkap")

# mencetak variabel pada string dengan format string
panjang = 10
lebar = 5
keliling = (2 * panjang) + (2 * lebar)
print("keliling dari (2 * %d) + (2 * %d) adalah %d" %(panjang, lebar, keliling))

```

Tugas

Anda diminta untuk membuat virtual environment ‘komputerteknik’ dan mencoba kode sederhana hello.py dan menjalankannya di VSCode yang menghasilkan output “Hello World” , seperti petunjuk pada modul ini.

Jawaban tugas ini adalah :

1. Screenshoot capture pada VScode program python yang anda ketik.
2. Screenshoot capture pada VScode hasil running program tsb pada window terminal VScode.

Hasil screenshoot bisa di rubah menjadi format pdf atau apapun sehingga sedemikian resolusi atau ukuran file total tidak melebihi 2MB seperti yang dipersyaratkan ketika mengupload jawaban anda.

Tips-tips jika ada error pada terminal saat eksekusi di Vscode

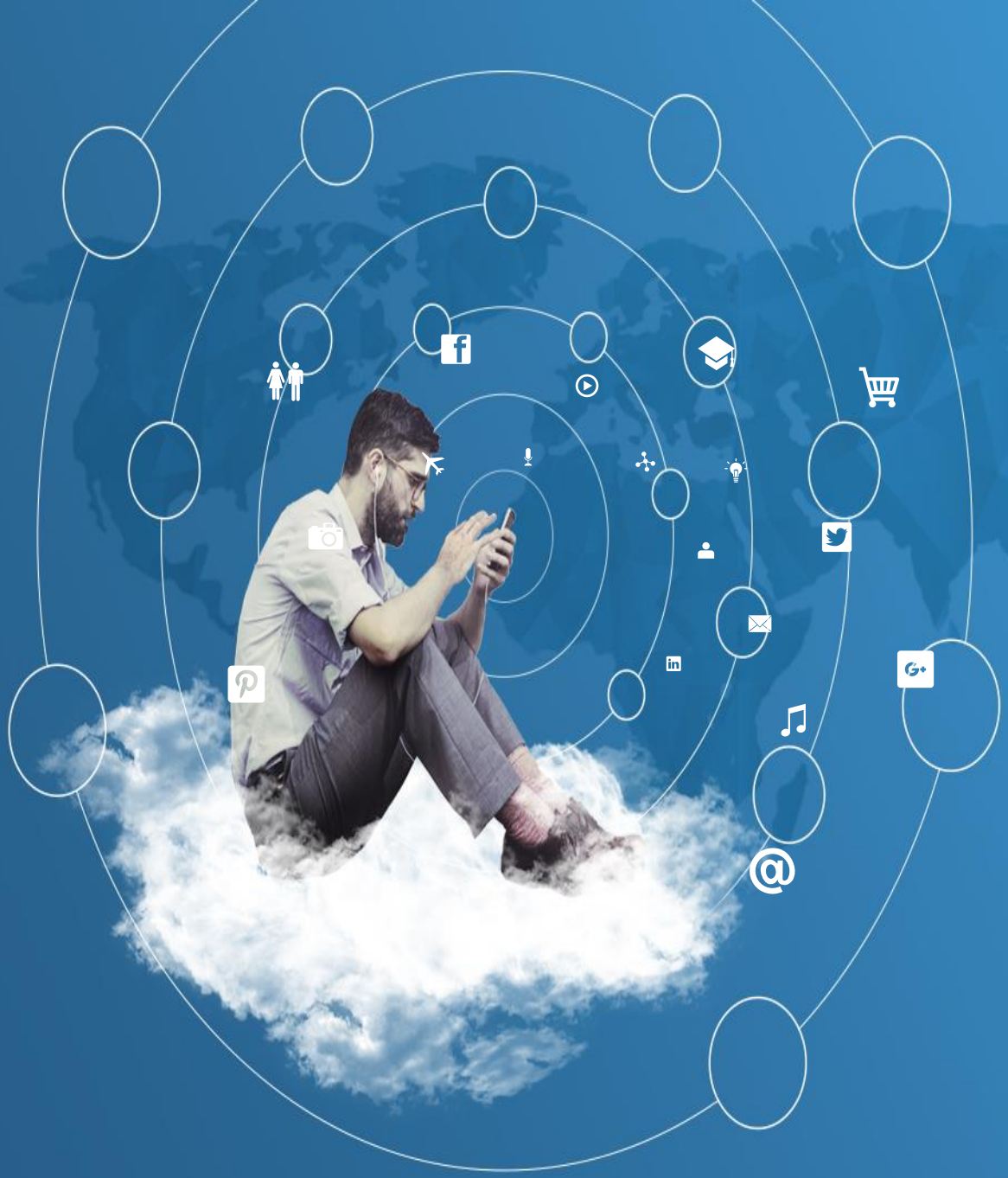
Error ini muncul jika menggunakan anaconda dan terminal Powershell pada Windows. Error terminal yang muncul :

```
CommandNotFoundError: Your shell has not been properly configured to use 'conda
activate'. If using 'conda activate' from a batch script, change your invocation
to 'CALL conda.bat activate'.
To initialize your shell, run
----------
----------
```

Tips Penanganan :

1. Tambahkan `Python.CondaPath` didalam setting Vscode. Tekan Ctrl+Shift+P → klik Preference: Open Settings (UI) → cari python:conda → pada field isian Python:Conda Path isikan path aplikasi conda.exe, misalnya : D:\Anaconda3\Scripts\conda.exe
2. Tangani masalah ini dari Powershell. Buka/jalankan terminal Powershell sebagai Administrator. Pada prompt yang muncul ketik **`>set-ExecutionPolicy RemoteSigned`**, Ketik Y [yes] & enter jika diminta konfirmasi. Sekarang Vscode dapat mengeksekusi program python dengan lancar.

Thank You



Programming for Science

Menerima Input dari User

Menerima Masukkan Data dari User

Menerima Masukan Data dengan Funtion “raw_input” dan “input”

- Dalam membuat sebuah program Anda membutuhkan fitur untuk meminta *input* dari *user*. Fitur tersebut berguna untuk menciptakan interaksi antara *user* dan *program* yang Anda bangun. Di Python untuk menerima *input* ada beberapa cara yang biasa digunakan.

Menerima Masukkan Data dari User

- **input**, *function* ini digunakan untuk menerima input sesuai dengan data yang diberikan oleh *user*.
- Di *input* Anda harus mengikuti aturan penulisan untuk memasukkan *input* dari tipe data tertentu.
- Sebagai contoh dibawah terdapat beberapa contoh aturan penulisan saat akan memberikan data dengan tipe data tertentu kepada input.

Menerima Masukkan Data dari User

```
# meminta input string nama
nama = input("Siapa nama anda : ")
print("Nama anda : ", nama)

# meminta input string name juga
print("What is your name?")
name = input()
print("Hello %s! how are you" % name)

# meminta input boolean : coba masukkan True
variabel_bool = input('masukkan data boolean : ')
print("isi variabel_bool : ", variabel_bool)

# meminta input float : coba masukkan 3.14
variabel_float = input('masukkan data float : ')
print("isi variabel_float : ", variabel_float)

# meminta input string : coba masukkan “lagi belajar python”
variabel_string = input('masukkan data string : ')
print("isi variabel_string : ", variabel_string)

# meminta input octal : coba masukkan 010
variabel_octal = input('masukkan data octal : ')
print("isi variabel_octal : ", variabel_octal)
```

Menerima Masukkan Data dari User - *lanjutan*

```
# meminta input hexa : coba masukkan 0x114
variabel_hexa = input('masukkan data hexa : ')
print("isi variabel_hexa : ", variabel_hexa)

# meminta input long : coba masukkan 12345678912345678 9L
variabel_long = input('masukkan data long : ')
print("isi variabel_long : ", variabel_long)

# meminta input dictionary : coba masukkan {'nama':'luffy',
'hobi':'makan', 'asal':'east blue'}
variabel_dict = input('masukkan data dictionary : ')
print("isi variabel_dict : ", variabel_dict)

# meminta input list : coba masukkan [1, 2, 3, 4, 5]
variabel_list = input('masukkan data list : ')
print("isi variabel_list : ", variabel_list)

# meminta input tuple : coba masukkan (1, 2, 3, 4, 5)
variabel_tuple = input('masukkan data tuple : ')
print("isi variabel_tuple : ", variabel_tuple)
```

Output dari Program adalah

```
Siapa nama anda : aryo
Nama anda :  aryo
What is your name?
aryo
Hello aryo! how are you
masukkan data boolean : True
isi variabel_bool :  True
masukkan data float : 3.14
isi variabel_float :  3.14
masukkan data string : dudy
isi variabel_string :  dudy
masukkan data octal : 0x114
isi variabel_octal :  0x114
masukkan data hexa : 276
isi variabel_hexa :  276
masukkan data long : 123456789345
isi variabel_long :  123456789345
masukkan data dictionary : {'nama':'Lutfy', 'hobi':''makan', 'asal':'east blue'}
isi variabel_dict :  {'nama':'Lutfy', 'hobi':''makan', 'asal':'east blue'}
masukkan data list : [1, 2, 3, 4, 5]
isi variabel_list :  [1, 2, 3, 4, 5]
masukkan data tuple : (1,2,3,4,5)
isi variabel_tuple :  (1,2,3,4,5)

(komputerteknik) D:\ECLIPSE-PYTHON\KomputerTeknik\Python_02_MenggunakanPython>
```

Hal Lain yang Harus Diingat dalam Penggunaan Python

Terdapat beberapa karakter khusus yang dinamakan escape character. Berikut adalah daftar beberapa escape character yang terdapat di Python :

Notasi Backslash	Karakter Hexadecimal	Penjelasan
\a	0x07	Bell atau alert
\b	0x08	Backspace
\cx		Control-x
\C-x		Control-x
\e	0x1b	Escape
\f	0x0c	Formfeed
\M-\C-x		Meta-Control-x
\n	0x0a	Newline
\nnn		Octal notation, dimana n berada di range 0.7
\r	0x0d	Carriage return
\s	0x20	Space
\t	0x09	Tab
\v	0x0b	Vertical tab
\x		Character x
\xnn		Notasi Hexadecimal, dimana n berada di range 0.9, a.f, atau A.F

Hal Lain yang Harus Diingat dalam Penggunaan Python

Pada kode terdapat sebuah simbol %s di dalam perintah print. Simbol tersebut dinamakan string formatter yang berfungsi untuk mencetak data sesuai dengan format yang diinginkan pada string yang disisipi simbol tersebut. Berikut adalah daftar beberapa string formatter yang disediakan Python:

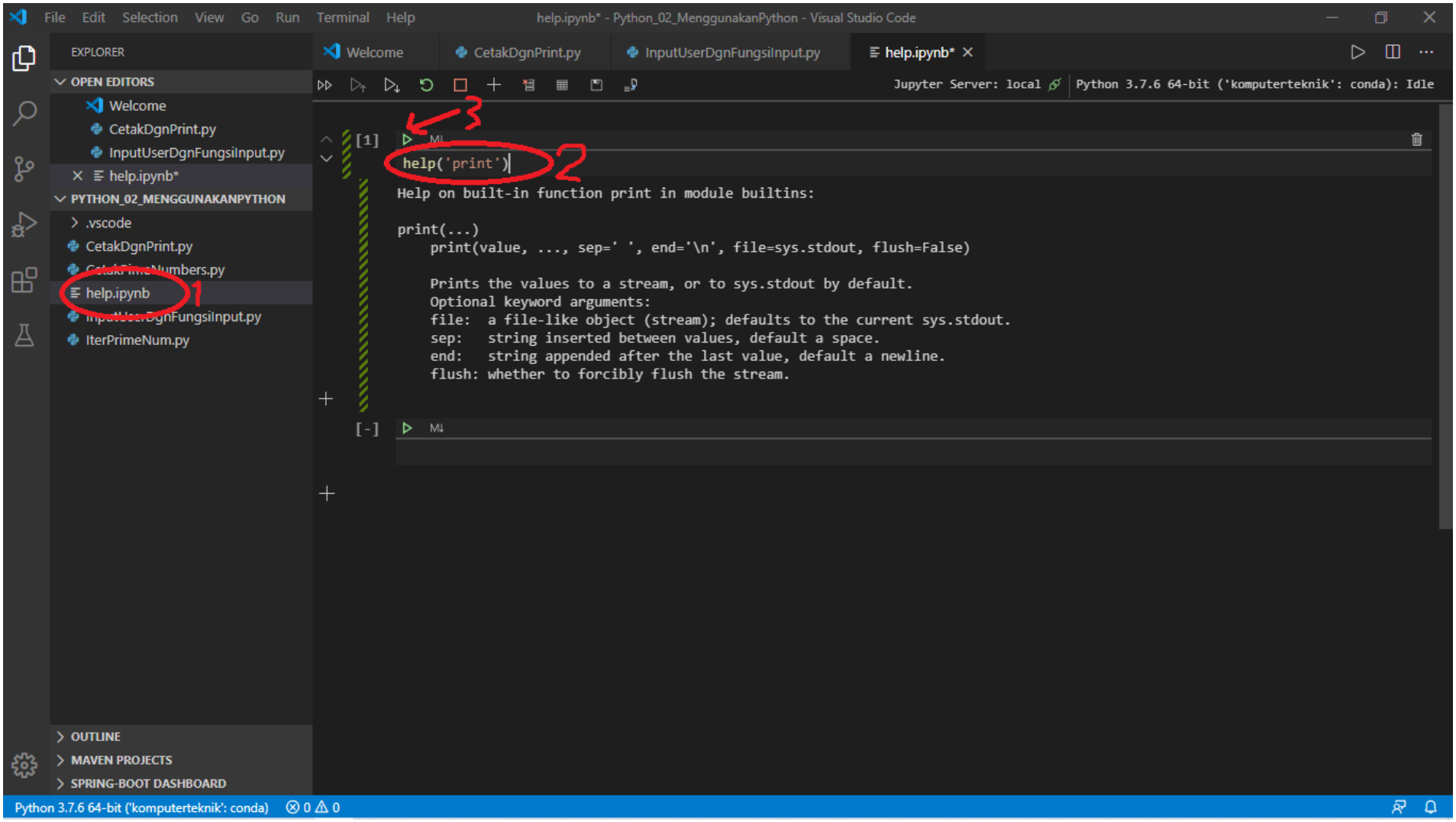
Operator	Penjelasan
%c	character
%s	Konversi string melalui str () sebelum memformat
%i	Dianggap sebagai bilangan bulat desimal
%d	Dianggap sebagai bilangan bulat desimal
%u	Unsigned decimal integer
%o	Bilangan bulat oktal
%x	Bilangan bulat heksadesimal (huruf kecil)
%X	Bilangan bulat heksadesimal (huruf besar)
%e	Notasi eksponensial (dengan huruf kecil ‘e’)
%E	Notasi eksponensial (dengan huruf besar ‘E’)
%f	Bilangan real floating point
%g	Yang lebih pendek dari% f dan% e
%G	Lebih pendek dari% f dan% E

Help sintaks Python

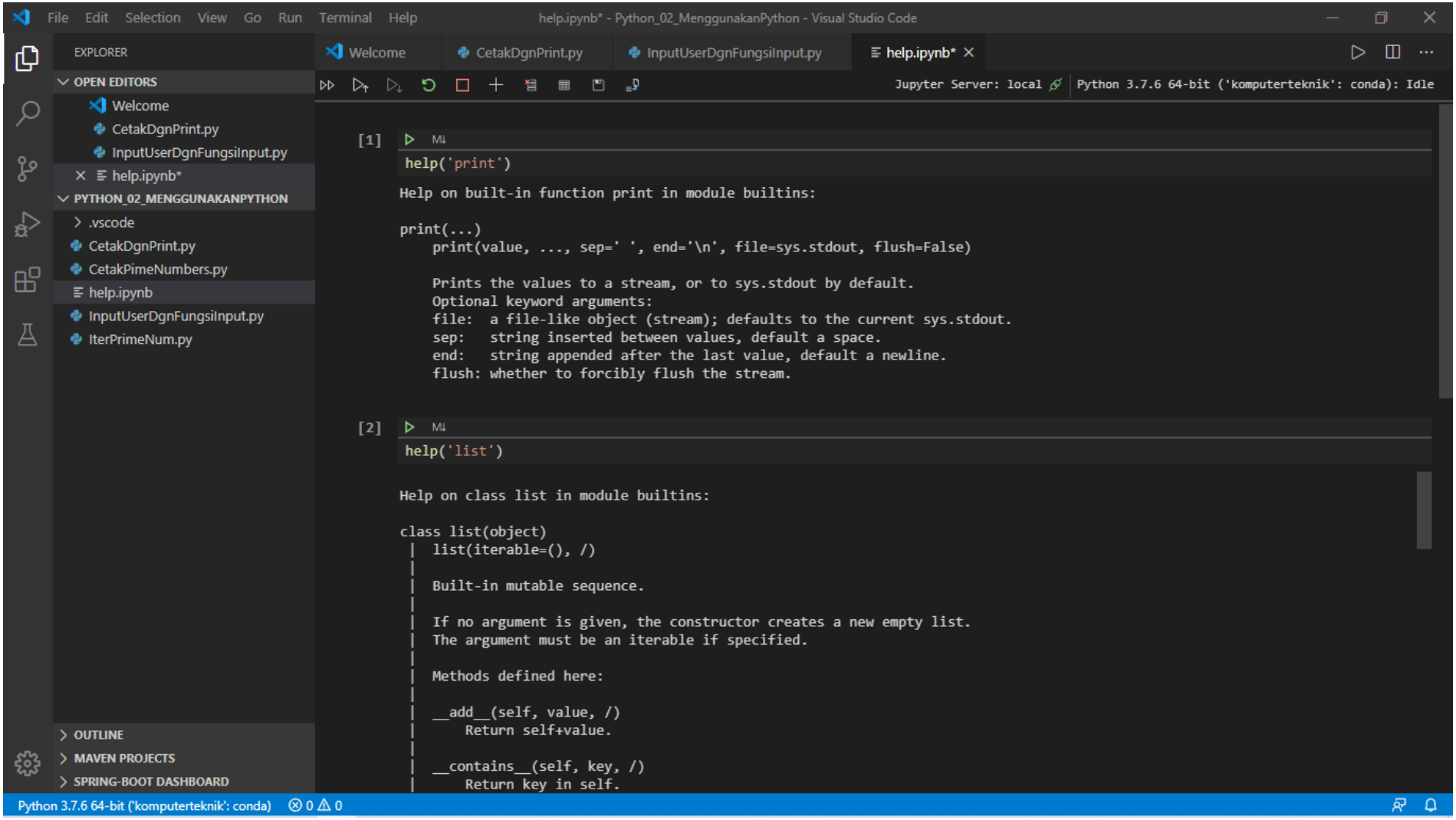
Informasi bantuan sintaks python :

- Pada dasarnya file program python ber-ekstensi *.py yg akan dieksekusi secara keseluruhan satu file tsb.
- Namun pada VS Code, dapat membuat file ber-ekstensi *.ipynb (misal help.ipynb). Yang bisa di eksekusi per bagian yang diinginkan.
- Klik *yes* jika VS Code mengeluarkan peringatan untuk menginstall ekstensi *jupiter note book*.
- Pada file ipynb tersebut ketik perintah :
  help('print') → bantuan sintaks print
  help('list') → bantuan sintaks list
  Dlsb ...

Help sintaks Python



Help sintaks Python



Tugas

Anda diminta untuk membuat/mengetik program seperti pada materi topik 2 pada halaman 4-5 ini pada IDE VScode.

Jawaban tugas ini adalah :

1. Screenshoot capture pada VScode program python yang anda ketik.
2. Screenshoot capture pada VScode hasil running program tsb pada window terminal VScode.

Hasil screenshoot bisa di rubah menjadi format pdf atau apapun sehingga sedemikian resolusi atau ukuran file total tidak melebihi 2MB seperti yang dipersyaratkan ketika mengupload jawaban anda.

Thank You

Programming for Science

Tipe Data dan Operator Python

Tipe Data di Python

- Variabel menyimpan data yang dilakukan selama program dieksekusi dan isinya dapat diubah oleh operasi – operasi tertentu pada program yang menggunakan variabel tersebut.
- Di dalam Python, terdapat beberapa tipe data yang cukup unik bila dibandingkan dengan bahasa pemrograman seperti C, Java, dan yang lainnya. Tipe data pada Python adalah sebagai berikut :

Tipe Data di Python

- Boolean, contoh True and False
- Complex, pasangan angka real dan imajiner, misalnya 1 + 5j
- Date, bilangan yang dapat dikonversi menjadi format tanggal, misalnya 26-09-2013
- Float, bilangan yang mempunyai koma, misalnya 3.14, 6.387
- Hexadecimal, bilangan dalam format heksa, misalnya 7b, 4d2
- Integer, bilangan bulat, misalnya 10, 20, 30, 15, 37
- Long, bilangan bulat yang panjang, misal 123456789123456789L
- None, data yang tidak terdefinisi tipe data apapun
- String, data yang berisi kalimat. Bisa dibentuk dengan diapit tanda ' dan ', atau diapit “ dan “, atau diapit “”” dan “”” untuk membentuk paragraf.
- List, sebuah data berupa untaian yang menyimpan berbagai tipe data dan isinya bisa diubah. Lebih lengkapnya akan dibahas di bab 6.
- Tuple, sebuah data berupa untaian yang menyimpan berbagai tipe data tapi isinya tidak bisa diubah. Lebih lengkapnya akan dibahas di bab 6.
- Dictionary, sebuah data berupa untaian yang menyimpan berbagai tipe data berupa pasangan penunjuk dan nilai. Lebih lengkapnya akan dibahas di bab 6.
- Objek, sebuah data yang berisi atribut dan method. Lebih lengkapnya akan dibahas di bab 10

Operator-operator di Python

- Selain variabel diatas, terdapat juga beberapa operator untuk pemrosesan data di Python. Agar lebih memahami seperti apa cara kerja operator di python, bukalah console python dan coba contoh disamping penjelasan tiap operator. Berikut operator yang ada di Python :

Operator-operator di Python

1. Aritmatika (Arithmetic Operator) :

Operator	Penjelasan	Contoh

+	Penjumlahan, menambahkan dua buah operan	a, b = 10, 5
		hasil = a + b
		# hasil akan 15
		print("hasil : ", hasil)

-	Pengurangan, mengurangka operan disebelahkiri operator dengan operan di sebelah kanan operator	a, b = 10, 8
		hasil = a – b
		# hasil akan 2
		print("hasil : ", hasil)

*	Perkalian, mengalikan operan di sebelah kiri dengan operan di sebelah kanan operator	a, b = 3, 5
		hasil = a * b
		# hasil akan 15
		print("hasil : ", hasil)

/	Pembagian, membagi operan di sebelah kiri dengan operan disebelah kanan operator	a, b = 4, 2
		hasil = a / b
		# hasil akan 2
		print("hasil : ", hasil)

Operator-operator di Python

1. Aritmatika (Arithmetic Operator) :

Operator	Penjelasan	Contoh

%	Modulus, mendapatkan sisa pembagian dari operan di sebelah kiri operator ketika dibagi oleh operan di sebelah kanan	a, b = 11, 2
		hasil = a % b
		# hasil akan 1
		print(“hasil : “, hasil)

**	Pemangkatan, memangkatkan operan disebelah kiri operator dengan operan di sebelah kanan operator	a, b = 11, 2
		hasil = a ** b
		# hasil akan 121
		print(“hasil : “, hasil)

//	Pembagian bulat, prosesnya sama seperti pembagian. Hanya saja angka dibelakang koma dihilangkan	a, b = 11, 2
		hasil = a // b
		# hasil akan 5
		print(“hasil : “, hasil)

Operator-operator di Python

2. Perbandingan (Comparison Operator) :

Operator	Penjelasan	Contoh

==	Memeriksa apakah kedua nilai (operan) yang dibandingkan sama atau tidak. Jika sama akan dikembalikan nilai True jika tidak sama akan dikembalikan nilai False.	a, b = 10, 10
		# hasil akan True
		print(“hasil : “, a == b)

!=	Memeriksa apakah kedua nilai yang dibandingkan sama atau tidak. Jika tidak sama akan dikembalikan nilai True jika sama akan dikembalikan nilai False.	a, b,= 10, 8
		# hasil akan True
		print(“hasil : “, a != b)
		c = 10
		# hasil akan False
		print(“hasil : “, a != c)

<>	Fungsinya sama dengan operator !=
	(operator ini berjalan pada versi python 2.x, tapi tidak dapat berjalan untuk versi python 3.x)
		a, b,= 10, 8
		# hasil akan True
		print(“hasil : “, a <> b)
		c = 10
		# hasil akan False
		print(“hasil : “, a <> c)

Operator-operator di Python

2. Perbandingan (Comparison Operator) :

Operator	Penjelasan	Contoh

>	Memeriksa apakah nilai di sebelah kiri operator lebih besar dari nilai di sebelah kanan operator	a, b = 4, 2
		# hasil akan True
		print("hasil : ", a > b)

<	Memeriksa apakah nilai di sebelah kiri operator lebih kecil dari nilai di sebelah kanan operator	a, b = 2, 4
		# hasil akan True
		print("hasil : ", a < b)

>=	Memeriksa apakah nilai di sebelah kiri operator lebih besar dari nilai di sebelah kanan operator atau memiliki nilai yang sama	a, b = 4, 2
		c = 4
		# hasil akan True
		print("hasil : ", a >= b)
		# hasil akan True
		print("hasil : ", a >=c)
		# hasil akan False
		print("hasil : ", b >= a)

Operator-operator di Python

2. Perbandingan (Comparison Operator) :

Operator	Penjelasan	Contoh

<=	Memeriksa apakah nilai di sebelah kiri operator lebih kecil dari nilai di sebelah kanan operator atau memiliki nilai yang sama	a, b = 4, 2
		c = 4
		# hasil akan False
		print(“hasil : “, a <= b)
		# hasil akan True
		print(“hasil : “, a <= c)
		# hasil akan True
		print(“hasil : “, b <= a)

Operator-operator di Python

3. Penugasan (Assignment Operator) :

Operator	Penjelasan	Contoh

=	Mengisikan nilai di sebelah kanan operator ke nilai di sebelah kiri operator	a = 10
		# hasil akan 10
		print(a)
		b = 15
		# hasil akan 15
		print(b)

+=	Menambahkan operan sebelah kiri operator dengan operan sebelah kanan operator kemudian hasilnya diisikan ke operan sebelah kiri	a, b,= 10, 8 # hasil akan 18 sama dgn a = a + b a += b print(“hasil : “, a)
-=	Mengurangi operan sebelah kiri operator dengan operan sebelah kanan operator kemudian hasilnya diisikan ke operan sebelah kiri	a, b,= 10, 8 # hasil akan 2 sama dgn a = a - b a -= b print(“hasil : “, a)

Operator-operator di Python

3. Penugasan (Assignment Operator) :

Operator	Penjelasan	Contoh
*=	Mengalikan operan sebelah kiri operator dengan operan sebelah kanan operator kemudian hasilnya diisikan ke operan sebelah kiri	a, b,= 10, 8 # hasil akan 80 sma dgn a = a * b a *= b print(“hasil : “, a)
/=	Membagi operan sebelah kiri operator dengan operan sebelah kanan operator kemudian hasilnya diisikan ke operan sebelah kiri	a, b,= 10, 5 # hasil akan 2 sama dgn a = a / b a /= b print(“hasil : “, a)
%=	Mengambil sisa bagi dari operan sebelah kiri operator dengan operan sebelah kanan operator kemudian hasilnya diisikan ke operan sebelah kiri	a, b,= 10, 4 # hasil akan 2 sama dgn a = a % b a %= b print(“hasil : “, a)

Operator-operator di Python

3. Penugasan (Assignment Operator) :

Operator	Penjelasan	Contoh
**=	Memangkatkan operan sebelah kiri operator dengan operan sebelah kanan operator kemudian hasilnya diisikan ke operan sebelah kiri	a, b,= 10, 2 # hasil akan 100 sma dgn a = a ** b a **= b print(“hasil : “, a)
//=	Membagi bulat operan sebelah kiri operator dengan operan sebelah kanan operator kemudian hasilnya diisikan ke operan sebelah kiri	a, b,= 10, 4 # hasil akan 2 sma dgn a = a // b a //= b print(“hasil : “, a)

Operator-operator di Python

4. Biner (Bitwise Operator) :

Operator	Penjelasan	Contoh

&	Operator biner AND, memeriksa apakah operan di sebelah kiri dan operan sebelah kanan mempunyai angka biner 1 di setiap bit. Jika keduanya bernilai 1 maka bit hasil operasi akan bernilai 1	a, b = 13, 37
		# a akan bernilai '0000 1101'
		# b akan bernilai '0010 0101'
		c = a & b
		# c akan bernilai 5 = '0000 0101'
		print(c)

|	Operator biner OR, memeriksa apakah operan di sebelah kiri dan operan sebelah kanan mempunyai angka biner 1 di setiap bit. Jika salah satunya bernilai 1 maka bit hasil operasi akan bernilai 1	a, b = 13, 37
		# a akan bernilai '0000 1101'
		# b akan bernilai '0010 0101'
		c = a | b
		# c akan bernilai 45 = '0010 1101'
		print(c)

^	Operator biner XOR, memeriksa apakah operan di sebelah kiri dan operan sebelah kanan mempunyai angka biner 1 di setiap bit. Jika keduanya bernilai 1 maka bit hasil operasi akan bernilai 0	a, b = 13, 37
		# a akan bernilai '0000 1101'
		# b akan bernilai '0010 0101'
		c = a ^ b
		# c akan bernilai 40 = '0010 1000'
		print(c)

Operator-operator di Python

4. Biner (Bitwise Operator) :

Operator	Penjelasan	Contoh

~	Operator biner Negative, membalik nilai bit. Misal dari 1 menjadi 0, dari 0 menjadi 1	a, b = 13, 37
		# a akan bernilai '0000 1101‘
		# b akan bernilai '0010 0101‘
		print(~a)
		print(~b)

<<	Operator penggeser biner ke kiri, deret bit akan digeser ke kiri sebanyak n kali	a, b = 13, 37
		# a akan bernilai '0000 1101'
		# b akan bernilai '0010 0101'
		# hasil bernilai 52 = ''0011 0100'
		print(a << 2)
		# hasil bernilai 148 = '1001 0100'
		print(b << 2)

>>	Operator penggeser biner ke kanan, deret bit akan digeser ke kanan sebanyak satu kali	a, b = 13, 37
		# a akan bernilai '0000 1101'
		# b akan bernilai '0010 0101'
		# hasil bernilai 3 = '0000 0011'
		print(a >> 2)
		# hasil bernilai 9 = '0000 1001'
		print(b >> 2)

Operator-operator di Python

5. Logika (Logical Operator) :

Operator	Penjelasan	Contoh

and	Jika kedua operan bernilai True, maka kondisi akan bernilai True. Selain kondisi tadi maka akan bernilai False	a, b = True, True
		# hasil akan True
		print(a and b)

or	Jika salah satu atau kedua operan bernilai True maka kondisi akan bernilai True. Jika keduanya False maka kondisi akan bernilai False	a, b = True, False
		# hasil akan True
		print(a or b)
		print(b or a)
		print(a or a)
		# hasil akan False
		print(b or b)

not	Membalikkan nilai kebeneran pada operan, misal jika asalnya True akan menjadi False dan begitupun sebaliknya	a, b = True, False
		
		# hasil akan False, True
		print(not a)
		print(not b)

Operator-operator di Python

6. Keanggotaan (Membership Operator) :

Operator	Penjelasan	Contoh
in	Memeriksa apakah nilai yang dicari berada pada list atau struktur data python lainnya. Jika nilai tersebut ada maka kondisi akan bernilai True	sebuah_list = [1, 2, 3,4 ,5] print(5 in sebuah_list)
not in	Memeriksa apakah nilai yang dicari tidak ada pada list atau struktur data python lainnya. Jika nilai tersebut tidak ada maka kondisi akan bernilai True	sebuah_list = [1, 2, 3,4 ,5] print(10 not in sebuah_list)

Operator-operator di Python

7. Identitas (Identity Operator) :

Operator	Penjelasan	Contoh

is	Memeriksa apakah nilai di sebelah kiri operan memiliki identitas memori yang sama dengan nilai di sebelah kanan operan. Jika sama maka kondisi bernilai True	a, b = 10, 10
		# hasil akan True
		print(a is b)

is not	Memeriksa apakah nilai di sebelah kiri operan memiliki identitas memori yang berbeda dengan nilai di sebelah kanan operan. Jika berbeda maka kondisi bernilai True	a, b = 10, 5
		# hasil akan True
		print(a is not b)

Prioritas Eksekusi Operator di Python

Dari sekian banyaknya operator yang telah disebutkan, masing – masing mempunyai prioritas pemroresan yang dapat dilihat pada tabel berikut. Prioritas tersebut makin ke bawah makin akhir untuk dieksekusi. Paling atas adalah yang akan didahulukan daripada operator lain, sedangkan paling bawah adalah operator yang paling terakhir dieksekusi :

Prioritas Eksekusi Operator di Python

Operator	Keterangan
**	Aritmatika
~, +, -	Bitwise
*, /, %, //	Aritmatika
+, -	Aritmatika
>>, <<	Bitwise
&	Bitwise
^, |	Bitwise
<=, <, >, >=	Perbandingan
<>, ==, !=	Perbandingan
=, %=, /=, //=, -=, +=, *=, **=	Penugasan
is, is not	identitas
in, not in	membership
not, or, and	logika

Tugas !!

- Lakukan percobaan pada VS Code berbagai operator di python (no.1 s/d 7) pada kolom contoh pada tabel
- Untuk lebih mudah eksekusi dan melihat hasilnya, anda bisa menggunakan mode ipynb (buat file xxx.ipynb) pada VS Code.
- Namun jika anda menggunakan mode py (file xxx.py) juga tidak bermasalah.

Selamat mencoba.

Tugas

Lakukan percobaan pada VS Code berbagai operator di python (no.1 s/d 7) pada kolom contoh pada tabel-tabel di modul ini.

Untuk lebih mudah eksekusi dan melihat hasilnya, anda bisa menggunakan mode ipynb (buat file xxx.ipynb) pada VS Code. Namun jika anda menggunakan mode py (file xxx.py) juga tidak bermasalah.

Jawaban tugas ini adalah :

1. Screenshoot capture pada VScode program python yang anda ketik.
2. Screenshoot capture pada VScode hasil running program tsb pada window terminal VScode.

Hasil screenshoot bisa di rubah menjadi format pdf atau apapun sehingga sedemikian resolusi atau ukuran file total tidak melebihi 2MB seperti yang dipersyaratkan ketika mengupload jawaban anda.

Thank You

Programming for Science

Membuat Pemilihan Kondisi (keyword “if”)

Penggunaan Operator Kondisional dan Logika pada Keyword “if”

- Dalam kehidupan sehari – hari pasti Anda menentukan pilihan untuk memulai sebuah aksi di pagi hari. “Kalau hari ini ga hujan saya akan main tenis”, “Kalau ada ongkos nanti nonton Man of Steel”, “Kalau ga hujan dan ada ongkos nanti mau pergi makan ramen”. Disadari atau tidak, pengandaian atau kondisional sudah menjadi bagian dari hidup Anda secara otomatis saat sebelum melakukan sebuah tugas.
- Dalam pemrograman pun demikian ada mekanisme dimana program akan menentukan aksi – aksi sesuai kondisi dari input atau nilai – nilai yang diproses selama program berjalan berlangsung. Pemilihan kondisi ini membutuhkan nilai “True” jika aksi yang diinginkan dibawah kondisi tersebut dieksekusi. Jika nilainya “False”, maka akan diperiksa kondisi lain yang sesuai atau akan langsung ke bagian program yang tidak memeriksa kondisi.

Penggunaan Operator Kondisional dan Logika pada Keyword “if”

- Di Python, terdapat beberapa keyword untuk membuat sebuah pemilihan kondisi. Ada if, elif, else. Tidak memerlukan kurawal atau penutup pada blok if tersebut. Sebuah statement akan dianggap blok if jika indentasinya lebih satu tab dari jumlah tab if diatasnya. Sebuah if akan diawali tanda titik dua baru dibawahnya terdapat kode program yang akan dieksekusi jika kondisi terpenuhi.
- Dalam membuat pemilihan kondisi Anda juga membutuhkan operator logika (and, not, or) dan perbandingan (==, <=, >=, >, <, <>, !=) untuk menyusun kondisi yang Anda butuhkan.

Penggunaan Operator Kondisional dan Logika pada Keyword “if”

- Berikut adalah contoh penggunaan if di Python. Contoh berikut menggunakan beberapa operator perbandingan untuk melihat hasil perbandingan dua buah angka. Dalam program berikut beberapa kondisi yang terpenuhi akan dieksekusi.
- Perlu diketahui bahwa Python menggunakan **identasi** pengetikkan program untuk menandakan **lingkup baris kode** terhadap baris yang indentasi-nya lebih maju. Lingkup yang lebih dalam akan dieksekusi lebih dulu setelah baris kode di atas nya

```
if username == username_from_db:
    if password == password_from_db:
        print("Username dan password cocok ")
    else:
        print("Password salah ")
else:
    print("User tidak terdaftar")
```

Penggunaan Operator Kondisional dan Logika pada Keyword “if”

Kode program kondisional1.py :

```
print("Masukkan dua buah angka")
print("Dan Anda akan check hubungan kedua angka tersebut")

angka1 = input("Masukkan angka pertama : ")
angka1 = int(angka1)

angka2 = input("Masukkan angka kedua : ")
angka2 = int(angka2)

if angka1 == angka2:
    print("%d sama dengan %d" % (angka1, angka2))

if angka1 != angka2:
    print("%d tidak sama dengan %d" % (angka1, angka2))

if angka1 < angka2:
    print("%d kurang dari %d" % (angka1, angka2))
```

Penggunaan Operator Kondisional dan Logika pada Keyword “if”

```
if angka1 > angka2:
    print("%d lebih dari %d" % (angka1, angka2))

if angka1 <= angka2:
    print("%d kurang dari sama dengan %d" % (angka1, angka2))

if angka1 >= angka2:
    print("%d lebih dari sama dengan %d" % (angka1, angka2))
```

Penggunaan Operator Kondisional dan Logika pada Keyword “if”

Output program :

```
Masukkan dua buah angka
Dan Anda akan check hubungan kedua angka tersebut
Masukkan angka pertama : 9
Masukkan angka kedua : 5
9 tidak sama dengan 5
9 lebih dari 5
9 lebih dari sama dengan 5
```

Penggunaan “else” pada “if”

- Keyword else digunakan dalam blok if untuk menampung berbagai kondisi yang berlawanan dengan kondisi pada if sebelumnya. Keyword else ini membutuhkan blok if atau elif diatasnya.
- Tanpa kedua keyword tadi, else tidak dapat digunakan. Berikut ini terdapat contoh penggunaan else, mari kita coba.

Penggunaan “else” pada “if”

Kode program kondisional2.py :

```
print("Masukkan dua buah angka..")
print("Dan Anda akan check hubungan kedua angka tersebut")

angka1 = input("Masukkan angka pertama : ")
angka1 = int(angka1)

angka2 = input("Masukkan angka kedua : ")
angka2 = int(angka2)

if angka1 == angka2 :
    print("%d sama dengan %d" % (angka1, angka2))
else:
    print("%d tidak sama dengan %d" % (angka1, angka2))
```

Penggunaan “else” pada “if”

Output program :

```
Masukkan dua buah angka..
Dan Anda akan check hubungan kedua angka tersebut
Masukkan angka pertama : 9
Masukkan angka kedua : 5
9 tidak sama dengan 5
```

Penggunaan “elif” pada “if”

- Jika pada kondisional1.py beberapa blok if akan dieksekusi, karena tidak ada pilihan lain pada masing – masing blok if. Pada contoh berikutnya beberapa if akan digabung dan membentuk sebuah blok if yang lebih besar karena adanya elif.
- Keyword elif ini berfungsi untuk membuat multi kondisional. Jadi jika kondisi di if paling atas tidak sesuai maka kondisi yang ada dibawahnya akan diperiksa dan jika cocok akan dieksekusi.
- Pada contoh berikutnya jika kondisi sudah sesuai pada blok teratas maka blok tersebutlah yang akan dieksekusi, berbeda dengan contoh pada kondisional1.py karena terdiri dari beberapa blok if yang dianggap berbeda oleh Python.
- Untuk lebih jelasnya mari coba kode berikut

Penggunaan “elif” pada “if”

Kode program kondisional3.py :

```
print("Masukkan dua buah angka..")
print("Dan Anda akan check hubungan kedua angka tersebut")

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

if angka1 == angka2:
    print("%d sama dengan %d" % (angka1, angka2))
elif angka1 != angka2:
    print("%d tidak sama dengan %d" % (angka1, angka2))
elif angka1 < angka2:
    print("%d kurang dari %d" % (angka1, angka2))
elif angka1 > angka2:
    print("%d lebih dari %d" % (angka1, angka2))
elif angka1 <= angka2:
    print("%d kurang dari sama dengan %d" % (angka1, angka2))
elif angka1 >= angka2:
    print("%d lebih dari sama dengan %d" % (angka1, angka2))
```

Penggunaan “elif” pada “if”

Output program kondisional3.py :

- Coba masukkan dengan angka 10 dan 20, maka blok if yang dieksekusi hanya blok kedua yang berisi kondisi angka1 tidak sama dengan angka2. Jelas berbeda dengan kode yang ada di kondisional1.py. Untuk lebih jelasnya coba perhatikan output program berikut

```
Masukkan dua buah angka..
Dan Anda akan check hubungan kedua angka tersebut
Masukkan angka pertama : 10
Masukkan angka kedua : 20
10 tidak sama dengan 20
```

- Pada output program diatas terlihat bahwa eksekusi program hanya pada elif angka1 =! angka2, langsung keluar dari lingkup if. Padahal kondisi diatas juga memenuhi angka1 < angka2, dan angka1 <= angka2. Hal ini berbeda pada program kondisional1.py .

Penggunaan “else” pada “if” (if bersarang)

- Misal ada sebuah kondisi seperti berikut, “Kalau saya punya uang saya akan pergi ke taman bermain, Lalu kalau uangnya cuma 10.000 cuma bakal naik komedi putar, kalau uangnya 20.000 bakal naik komedi putar dan bom bom car”.
- Jika Anda perhatikan setelah kondisi pertama ada kondisi lagi yang masih berada dibawah kondisi pertama. Kondisi semacam ini dapat disebut dengan kondisi bersarang (nested if).
- Di Python, untuk membuat sebuah blok if di dalam if, maka blok if yang ingin disimpan di dalam sebuah if harus mempunyai satu tab lebih dibanding if sebelumnya. Anda dapat membuat if bersarang di dalam if bersarang hingga tingkat sedalam yang Anda inginkan

Penggunaan “else” pada “if” (if bersarang)

Agar lebih paham, coba Kode program kondisional4.py :

```
username = input("masukkan username : ")
password = input("masukkan password : ")

username_from_db = "user"
password_from_db = "admin"

if username == username_from_db:
    if password == password_from_db:
        print("Username dan password cocok ")
    else:
        print("Password salah ")
else:
    print("User tidak terdaftar")
```

Penggunaan “else” pada “if” (if bersarang)

Pada contoh diatas, Anda diminta masukan berupa “username” dan “password”.

Kemudian ada sebuah variabel yang diasumsikan mengambil data “username” dan “password” dari database.

Blok if akan memeriksa apakah user sudah sesuai atau belum, jika tidak sesuai maka akan ditampilkan “User tidak terdaftar”. Jika “username” sesuai maka kondisi selanjutnya adalah memeriksa “password” jika sesuai maka akan muncul notifikasi “Username dan password cocok”, jika tidak sesuai maka akan muncul notifikasi “Password salah”. Lebih jelasnya perhatikan gambar berikut :

Penggunaan “else” pada “if” (if bersarang)

Output Kode program kondisional4.py :

```
masukkan username : bejo
masukkan password : bejo
User tidak terdaftar

masukkan username : user
masukkan password : bejo
Password salah

masukkan username : bejo
masukkan password : admin
User tidak terdaftar

masukkan username : user
masukkan password : admin
Username dan password cocok
```

Tugas

Lakukan percobaan pada VS Code untuk Program kondisional1.py, kondisional2.py, kondisional3.py, dan kondisional4.py.

Jawaban tugas ini adalah :

1. Screenshoot capture pada VScode program python yang anda ketik.
2. Screenshoot capture pada VScode hasil running program tsb pada window terminal VScode.

Hasil screenshoot bisa di rubah menjadi format pdf atau apapun sehingga sedemikian resolusi atau ukuran file total tidak melebihi 2MB seperti yang dipersyaratkan ketika mengupload jawaban anda.

Thank You

Programming for Science

Menyusun Pengulangan

Mengenal Pengulangan “for” dan “while”

- Seperti pada bahasa pemrograman lainnya, Python memiliki mekanisme pengulangan untuk menjalankan pekerjaan – pekerjaan yang berulang. Pada umumnya pengulangan terdiri dua jenis.
- Ada pengulangan yang terbatas dan tidak terbatas. Pengulangan terbatas biasanya dilakukan pengulangan dari 1 hingga kesekian kali (n kali). Perulangan terbatas menggunakan for.
- Sedangkan pengulangan tidak terbatas menggunakan while. Dalam pengulangan menggunakan while pengulangan akan terus dilakukan selama kondisional dalam perulangan while tetap dalam keadaan true jika dalam keadaan false maka pengulangan while akan berhenti.

Menyusun Pengulangan dengan “for”

- Kita melangkah ke pengulangan yang terbatas dulu yah. Dalam pengulangan for, tidak seperti di bahasa pemorgraman C atau Java yang menggunakan nilai incremental untuk melakukan pengulangan.
- Di Python, for melakukan pengulangan dengan meng-iterasi elemen dari sebuah list.
- List ini dapat berisi kumpulan karakter, kumpulan string, kumpulan angka, atau kumpulan data jenis lainnya yang disediakan Python. (Untuk lebih lengkapnya di bab berikutnya akan dibahas lebih jauh tentang List di Python).

Menyusun Pengulangan dengan “for”

- Misal disini ada sebuah list yang berisi [1, 2, 3, 4, 5], (sebuah list diawali oleh tanda '[' dan ditutup oleh tanda ']').
- Banyaknya elemen pada list tersebut menentukan banyaknya pengulangan yang akan dilakukan saat melakukan pengulangan. Mari kita lihat implementasinya pada kode dibawah ini :

Menyusun Pengulangan dengan “for”

Kode program pengulangan1.py :

```
# pengulangan sebanyak 5 kali
for i in [1, 2, 3, 4, 5]:
    print("Ini pengulangan ke - ", i)
```

Pada contoh diatas, akan dicetak teks “ini pengulangan ke - “ sebanyak 5 kali. Nilai 'i' pada pengulangan tersebut akan selalu berganti nilainya setiap tahap pengulangan dilakukan. Misal ketika pengulangan pertama, nilai 'i' akan berisi 1, ketika pengulangan kedua, nilai 'i' akan berisi 2, begitu seterusnya sampai elemen terakhir. Jika kode diatas dieksekusi akan menampilkan hasil seperti pada gambar dibawah ini :

Menyusun Pengulangan dengan “for”

Output program pengulangan1.py :

```
Ini pengulangan ke -  1
Ini pengulangan ke -  2
Ini pengulangan ke -  3
Ini pengulangan ke -  4
Ini pengulangan ke -  5
```

Menyusun Pengulangan dengan “for”

- Selain menggunakan list yang berisi angka, List yang berisi string dapat digunakan juga untuk melakukan pengulangan for di Python.
- Misal terdapat list yang berisi seperti berikut [“Rawon”, “Nasi Kuning”, “Soto Madura”, “Kupat Tahu”, “Kerak Telor”, “Rendang Batoko”, “Pempek Selam”, “Ayam Betutu”], dalam list tersebut terdapat elemen sebanyak delapan jenis masakan nusantara.
- Dengan demikian ketika pengulangan for menggunakan list masakan tadi, pengulangan akan dijalankan sebanyak delapan kali. Mari Anda lihat implementasinya pada kode dibawah ini :

Menyusun Pengulangan dengan “for”

Kode program pengulangan2.py :

```
# pengulangan sebanyak 8 kali
for i in ["Rawon", "Nasi Kuning", "Soto Madura", "Kupat Tahu", "Kerak Telor",
"Rendang Batoko", "Pempek Selam", "Ayam Betutu"]:
    print(i, " adalah masakan khas nusantara")
```

Kode diatas jika dieksekusi akan terlihat seperti dibawah ini :

```
Rawon  adalah masakan khas nusantara
Nasi Kuning  adalah masakan khas nusantara
Soto Madura  adalah masakan khas nusantara
Kupat Tahu  adalah masakan khas nusantara
Kerak Telor  adalah masakan khas nusantara
Rendang Batoko  adalah masakan khas nusantara
Pempek Selam  adalah masakan khas nusantara
Ayam Betutu  adalah masakan khas nusantara
```

Menyusun Pengulangan dengan “for”

- Sekarang Anda coba contoh terakhir dengan menggunakan string.
- String pada dasarnya merupakan list karakter.
- Misal terdapat string seperti berikut “abcde”. Jika string tersebut digunakan pada pengulangan for, maka akan terjadi pengulangan sebanyak lima kali.
- Coba lihat kode dibawah ini :

Menyusun Pengulangan dengan “for”

Kode program pengulangan3.py :

```
# pengulangan sebanyak 5 kali
for i in "abcde":
    print(i, " adalah alfabet")
```

Kode diatas jika dieksekusi akan terlihat seperti dibawah ini :

```
a  adalah alfabet
b  adalah alfabet
c  adalah alfabet
d  adalah alfabet
e  adalah alfabet
```

Memahami Function “range”

- Kalau diperhatikan list yang dipakai pada pengulangan1.py, pengulangan tersebut menggunakan list angka yang sudah jadi atau di-hardcode.
- Nah bagaimana nih kalau ingin membentuk suatu pola atau ingin membuat list incremental agar pengulangan for di Python ini mirip di Java atau C.
- Di Python terdapat fungsi yang bernama **range**. **Range** ini menghasilkan deret angka dengan parameter (start, stop, step).
- Start adalah batas awal dari list, stop adalah batas akhir dari list, step adalah jarak antar angka yang dihasilkan oleh **range**.
- Ada beberapa kasus penting yang perlu diperhatikan saat menggunakan range. Coba perhatikan kode dibawah ini :

Memahami Function “range”

Kode program pengulangan4.py :

```
# kasus - 1 : jika step tidak disertakan maka step akan diisi 1 secara default
print(list(range(1, 10)))

# kasus - 2 : jika step disertakan maka step akan sesuai dengan angka yang diisikan
print(list(range(1, 10, 2)))
print(list(range(1, 10, 3)))
print(list(range(1, 10, 4)))
print(list(range(1, 10, 5)))

# kasus - 3 : jika step melebihi stop maka list hanya akan berisi start
print(list(range(1, 10, 11)))

# kasus - 4 : jika start lebih besar nilainya daripada stop maka list akan kosong
print(list(range(10, 1)))

# kasus - 5 : jika start lebih besar nilainya daripada stop dan
# jika step melebihi stop maka list akan kosong
print(list(range(10, 1, 2)))
print(list(range(10, 1, 11)))
```

Memahami Function “range”

Kode program pengulangan4.py :

```
# kasus - 6 : jika start lebih besar daripada stop dan step bernilai minus
# dan jika start dikurangi step menghasilkan angka positif
# maka list akan berisi deret angka menurun
print(list(range(10, 1, -1)))

# kasus - 7 : jika start lebih besar daripada stop dan step bernilai minus
# dan jika start dikurangi step bernilai minus maka list hanya akan berisi start
print(list(range(10, 1, -11)))

# kasus - 8 : jika step bernilai 0 maka akan terjadi error
print(list(range(1, 10, 0)))

# kasus -
9 : jika start lebih besar daripada stop dan step bernilai 0 maka akan terjadi erro
r
print(list(range(10, 1, 0)))
```

Memahami Function “range”

Output program pengulangan4.py :

```
[1, 2, 3, 4, 5, 6, 7, 8, 9]
[1, 3, 5, 7, 9]
[1, 4, 7]
[1, 5, 9]
[1, 6]
[1]
[]
[]
[]
[10, 9, 8, 7, 6, 5, 4, 3, 2]
[10]
Traceback (most recent call last):
  File "d:/ECLIPSE-
PYTHON/PythonBasics/Python_05_MenyusunPengulangan/pengulangan4.py", line 31, in
<module>
    print(list(range(1, 10, 0)))
ValueError: range() arg 3 must not be zero
```

Menggunakan Function “range” pada Pengulangan “for”

- Bagaimana dengan pengenalan range diatas ? Mudah kan ?
- Nah sekarang Anda akan coba menggunakan range dalam pengulangan for. Coba perhatikan contoh program pengulangan5.py berikut :

```
# mencetak sembilan kali
for i in range(1, 10):
    print("ini pengulangan ke - ", i)
```

Output program pengulangan5.py :

```
ini pengulangan ke -  1
ini pengulangan ke -  2
ini pengulangan ke -  3
ini pengulangan ke -  4
ini pengulangan ke -  5
ini pengulangan ke -  6
ini pengulangan ke -  7
ini pengulangan ke -  8
ini pengulangan ke -  9
```

Menggunakan Function “range” pada Pengulangan “for”

- Pada contoh diatas akan terjadi pengulangan sebanyak 9 kali terhadap statement dibawah for.
- Dengan menggunakan range, Anda tidak perlu repot untuk membuat list terlebih dahulu untuk menentukan banyaknya pengulangan yang akan Anda lakukan terhadap *statement for*.
- Agar lebih memahami lagi pengulangan for, kita coba lagi pelajari dua contoh berikut. Berikut ada kasus (1) membuat sebuah segitiga yang dibuat dari kumpulan bintang dan; (2) membuat baris bilangan prima. Untuk lebih jelasnya coba perhatikan dua kasus berikut :

Menggunakan Function “range” pada Pengulangan “for”

Kode program pengulangan6.py, mencetak segitiga bintang.

```
for i in range(0, 10):
    for j in range(0, i+1):
        if j == i:
            print("x")
        else:
            print("*", end='')
print("")
```

Output program pengulangan6.py :

```
x
*x
**x
***x
****x
*****x
******x
*******x
********x
*********x
```

Menggunakan Function “range” pada Pengulangan “for”

Kode program pengulangan7.py, mencari bilangan prima. Bilangan prima adalah bilangan yang hanya bisa dibagi 1 dan bilangan itu sendiri.

```
for i in range(1, 20):
    count_zero_remainder = 0
    for j in range(1, i+1):
        num_remainder = i % j
        # print num_remainder,
        if num_remainder == 0:
            count_zero_remainder = count_zero_remainder + 1

    if count_zero_remainder == 2:
        print(i, " adalah bilangan prima")
    else:
        print(i, " bukanlah bilangan prima")
```

Menggunakan Function “range” pada Pengulangan “for”

Output program pengulangan7.py

```
1  bukanlah bilangan prima
2  adalah bilangan prima
3  adalah bilangan prima
4  bukanlah bilangan prima
5  adalah bilangan prima
6  bukanlah bilangan prima
7  adalah bilangan prima
8  bukanlah bilangan prima
9  bukanlah bilangan prima
10  bukanlah bilangan prima
11  adalah bilangan prima
12  bukanlah bilangan prima
13  adalah bilangan prima
14  bukanlah bilangan prima
15  bukanlah bilangan prima
16  bukanlah bilangan prima
17  adalah bilangan prima
18  bukanlah bilangan prima
19  adalah bilangan prima
```

Menyusun Pengulangan dengan “while”

- Sekarang kita akan bahas pengulangan yang menggunakan while.
- Pengulangan while memiliki cara kerja selama kondisi tertentu bernilai true maka pengulangan akan diteruskan sampai kondisi bernilai false.
- Tentunya dalam kondisi yang dipakai untuk eksekusi while memerlukan operator logika dan perbandingan seperti yang sudah di jelaskan di bab 3.

Menyusun Pengulangan dengan “while”

Kode berikut dieksekusi apabila variabel “angka” masih dibawah 10. Kode pengulangan8.py

```
angka = 0
while (angka < 10):
    print("Aku sudah berjalan sebanyak ", angka, " langkah ")
    angka += 1
```

Output program pengulangan8.py :

```
Aku sudah berjalan sebanyak  0  langkah
Aku sudah berjalan sebanyak  1  langkah
Aku sudah berjalan sebanyak  2  langkah
Aku sudah berjalan sebanyak  3  langkah
Aku sudah berjalan sebanyak  4  langkah
Aku sudah berjalan sebanyak  5  langkah
Aku sudah berjalan sebanyak  6  langkah
Aku sudah berjalan sebanyak  7  langkah
Aku sudah berjalan sebanyak  8  langkah
Aku sudah berjalan sebanyak  9  langkah
```

Menyusun Pengulangan dengan “while”

Pada contoh diatas kondisi untuk melakukan pengulangan ditaruh di while. Sekarang Anda coba taruh kondisi pengulangan di dalam pengulangannya. Kode pengulangan9.py

```
terus_tanya = True
while terus_tanya:
    temp = input('masukkan angka kurang dari 10 !! : ')
    angka = int(temp)
    if angka < 10:
        terus_tanya = False
    else:
        terus_tanya = True
```

Output program pengulangan9.py :

```
masukkan angka kurang dari 10 !! : 45
masukkan angka kurang dari 10 !! : 13
masukkan angka kurang dari 10 !! : 7

(komputerteknik) D:\ECLIPSE-
PYTHON\PythonBasics\Python_05_MenyusunPengulangan>
```

Menyusun Pengulangan dengan “while”

Untuk memahami pengulangan while lebih lanjut, berikut terdapat contoh penjumlahan angka dari 1 sampai 10. Dalam pengulangan ini terdapat sebuah variabel jml_angka yang berfungsi untuk menampung angka – angka yang akan ditambahkan dengan angka berikutnya di setiap pengulangan.

Kode pengulangan10.py

```
i = 1
jml_angka = 0
while i <= 10:
    print('loop ke - %d : %d + %d' % (i, jml_angka, i))
    jml_angka = jml_angka + i
    i += 1
print('total angka yang dijumlahkan : ', jml_angka)
```

Menyusun Pengulangan dengan “while”

Output program pengulangan10.py :

```
loop ke - 1 : 0 + 1
loop ke - 2 : 1 + 2
loop ke - 3 : 3 + 3
loop ke - 4 : 6 + 4
loop ke - 5 : 10 + 5
loop ke - 6 : 15 + 6
loop ke - 7 : 21 + 7
loop ke - 8 : 28 + 8
loop ke - 9 : 36 + 9
loop ke - 10 : 45 + 10
total angka yang dijumlahkan :  55
```

Tugas

Lakukan percobaan pada VS Code untuk Program pengulangan3.py, pengulangan4.py, pengulangan7.py, dan pengulangan9.py.

Jawaban tugas ini adalah :

1. Screenshoot capture pada VScode program python yang anda ketik.
2. Screenshoot capture pada VScode hasil running program tsb pada window terminal VScode.

Hasil screenshoot bisa di rubah menjadi format pdf atau apapun sehingga sedemikian resolusi atau ukuran file total tidak melebihi 2MB seperti yang dipersyaratkan ketika mengupload jawaban anda.

Thank You

Programming for Science

Mengenal Data Struktur Python Tingkat Lanjut

Mengenal List, Dictionary dan Tuple

- Data yang kompleks biasanya dapat menampung objek atau entitas yang banyak.
- Misal ada sebuah toples yang berisi banyak permen dengan ragam jenisnya. Dalam sebuah toples dapat saja ada permen rasa asem, permen rasa stroberi, permen rasa nanas, permen rasa asem lagi, dan rasa lainnya.
- Tapi bukan hanya menyimpan entitas yang banyak saja, struktur data pun ada yang menyimpan jaringan sebuah graf, antrian, tumpukan, dan banyak lagi tipe lainnya.

Mengenal List, Dictionary dan Tuple

- Di Python sendiri, terdapat beberapa tipe struktur data yang dapat digunakan oleh penggunanya.
- Tapi tiga tipe data yang umum digunakan, diantaranya list, tuple, dan dictionary.
- Ketiga tipe data struktur ini sangat membantu sekali bagi programmer Python yang membutuhkan tipe data banyak.
- Ketiga tipe data ini adalah objek Python yang berarti jika Anda mendefinisikan struktur data tersebut, Anda dapat menggunakan method – method yang berhubungan dengan pengolahan struktur data tersebut.
- Selain itu terdapat pula function untuk mengolah tiga struktur data tadi seperti mencari nilai max, min, hitung panjang, dan perbandingan isi.

Mengenal List, Dictionary dan Tuple

- Untuk mendefinisikan sebuah list Anda cukup buat sebuah variabel dengan nama yang diinginkan, kemudian isi variabel tersebut dengan list yang akan dibuat.
- List diawali dengan tanda '[' dan diakhiri dengan tanda ']'. Isinya dapat beragam, dapat string, number, object, bahkan list lagi.
- Pemisah antara data yang satu dengan yang lainnya digunakan tanda koma.
- List dapat ditambah isinya, dirubah data pada elemennya, hapus elemen, dan hapus seluruh list.

Mengenal List, Dictionary dan Tuple

- Hampir sama dengan list, tuple diawali dengan tanda '(' dan ')'.
- Elemen pada tuple tidak dapat dihapus dan diubah. Maka dari itu tuple disebut *immutable sequence*.
- Lebih jelasnya nanti Anda akan lihat perbedaan tuple dengan list.

Mengenal List, Dictionary dan Tuple

- Beda halnya antara dengan list dan tuple, pada dictionary setiap data akan memiliki pengenalnya masing – masing.
- Pengenal tersebut dinamakan dengan *key* dan datanya dinamakan dengan *value*.
- Dictionary diawali dengan tanda '{' dan diakhiri dengan tanda '}'.
- Khusus untuk *key* pada dictionary, nilainya harus berupa tipe data yang tidak dapat diganti seperti tuple, string dan number. Tapi umumnya *key* berisi number dan string.
- Karena jika Anda mencoba memasukkan tipe data yang mutable, akan keluar peringatan 'unhashable type' saat mendefinisikan dictionary yang *key*-nya berupa tipe data mutable.

Mengenal List, Dictionary dan Tuple

Agar lebih paham mari Anda coba mendefinisikan list, tuple, dan dictionary kedalam sebuah variabel. Kode program strukdat1.py

```
# cara mendefinisikan list
sebuah_list = ['Zorin OS', 'Ubuntu', 'FreeBSD', 'NetBSD',
               'OpenBSD', 'Backtrack', 'Fedora', 'Slackware']

# cara mendefinisikan tuple
sebuah_tuple = (0, 1, 2, 3, 4, 5, 6, 7, 8, 9)

# cara mendefinisikan dictionary
sebuah_dictionary = {'nama': 'Wiro Sableng',
                     'prodi': 'ilmu komputer',
                     'email': 'wirosableng@localhost',
                     'website': 'http://www.sitampanggarang.com'
                     }

print(sebuah_list)
print(sebuah_tuple)
print(sebuah_dictionary)
```

Mengenal List, Dictionary dan Tuple

Dengan menggunakan perintah print maka Anda dapat mencetak isi list, tuple, atau dictionary yang hasil keluarannya berupa string. Output program strukdat1.py

```
['Zorin OS', 'Ubuntu', 'FreeBSD', 'NetBSD', 'OpenBSD', 'Backtrack',
'Fedora', 'Slackware']
(0, 1, 2, 3, 4, 5, 6, 7, 8, 9)
{'nama': 'Wiro Sableng', 'prodi': 'ilmu komputer', 'email':
'wirosableng@localhost', 'website':
'http://www.sitampanggarang.com'}
```

Cara Akses List, Tuple dan Dictionary

- Setelah Anda mengenal cara membuat ketiga struktur data tersebut, sekarang Anda coba cara mengakses elemen – elemen pada struktur data tersebut.
- Ada beberapa cara mengakses elemen pada struktur data tersebut yang biasa dilakukan misalnya mengakses salah satu elemen dengan menggunakan indeks, slicing indeks, dan mengakses elemen lewat pengulangan.

Cara Akses List, Tuple dan Dictionary

- Untuk akses elemen lewat indeks elemen Anda panggil nama list kemudian disusul indeks elemen yang Anda inginkan dengan diapit tanda baca “[“ dan “]”, misal ada sebuah list dengan nama daftar_barang kemudian ingin mengakses indeks ke – 10, maka pemanggilan indeks pada list tersebut adalah daftar_barang[9].
- Kenapa di pemanggilan indeks nya dari 9 ? karena indeks tersebut diawali dari 0 sehingga indeks yang diinginkan akan dikurangi 1.
- Begitupun dengan tuple cara akses salah satu elemenya sama dengan cara akses salah satu elemen di list. Pada dictionary, untuk mengakses salah satu elemennya Anda panggil salah satu *key*-nya untuk mendapatkan data yang ditunjuk *key* tersebut.

Cara Akses List, Tuple dan Dictionary

- Slicing indeks merupakan cara untuk mengakses beberapa elemen pada list dan tuple. Cara ini tidak dapat dilakukan di dictionary.
- Slicing indeks dilakukan dengan memanggil list atau tuple kemudian tentukan indeks awal slicing dan batas akhirnya. Kemudian indeks tersebut dipisahkan dengan tanda “:” dan diapit oleh tanda “[“ dan “]”.
- Misal ada sebuah list daftar_barang kemudian ingin mengambil 10 datanya dari indeks ke – 2 maka pemanggilannya adalah daftar_barang[1:11].

Cara Akses List, Tuple dan Dictionary

- Berikutnya cara terakhir yang biasa dilakukan untuk mengakses elemen secara keseluruhan adalah dengan melalui pengulangan for.
- Melalui cara tersebut, isi dari list, tuple, dan dictionary dapat diambil elemennya selama iterasi pada pengulangan for.
- Pada list dan tuple jika datanya diambil lewat pengulangan for, setiap elemen akan langsung diekstrak di setiap iterasi dan dapat digunakan untuk keperluan pemrosesan pada proses di setiap iterasi.
- Kalau pada dictionary di setiap iterasi pengulangan bukan elemen yang diektrak tapi *key*-nya. Jadi saat ingin mengambil datanya Anda harus memanggil elemen dictionary tersebut dengan *key* yang didapatkan disetiap iterasi.

Akses List, Tuple dan Dictionary

Dibawah ini adalah contoh mengakses elemen dari list, tuple dan dictionary dengan tiga cara yang biasa dipakai programmer python. **Kode program strukdat2.py**

```
# cara mendefinisikan list
sebuah_list = ['Zorin OS',
               'Ubuntu',
               'FreeBSD',
               'NetBSD',
               'OpenBSD',
               'Backtrack',
               'Fedora',
               'Slackware']

# cara mendefinisikan tuple
sebuah_tuple = (0, 1, 2, 3, 4, 5, 6, 7, 8, 9)

# cara mendefinisikan dictionary
sebuah_dictionary = {'nama': 'Wiro Sableng',
                     'prodi': 'ilmu komputer',
                     'email': 'wirosableng@localhost',
                     'website': 'http://www.sitampanggarang.com'
                     }
```

Akses List, Tuple dan Dictionary

Kode program strukdat2.py

```
# mengakses elemennya
print("mengakses salah satu elemen : ")
print("-----------------------------")
print(sebuah_list[5])
print(sebuah_tuple[8])
print(sebuah_dictionary['website'])

print("\n\n")

# mengakses beberapa elemen
print("mengakses beberapa elemen : ")
print("-----------------------------")
print(sebuah_list[2:5])
print(sebuah_tuple[3:6])

print("\n\n")
```

Akses List, Tuple dan Dictionary

Kode program strukdat2.py

```
# mengakses elemennya dengan looping
print("mengakses semua elemen dengan looping for : ")
print("----------------------------")

for sebuah in sebuah_list:
    print(sebuah, end='')
print("\n")

for sebuah in sebuah_tuple:
    print(sebuah, end='')
print("\n")

for sebuah in sebuah_dictionary:
    print(sebuah, ':', sebuah_dictionary[sebuah], end='')
```

Akses List, Tuple dan Dictionary

Output program strukdat2.py

```
mengakses salah satu elemen :
----------------------------
Backtrack
8
http://www.sitampanggarang.com

mengakses beberapa elemen :
----------------------------
['FreeBSD', 'NetBSD', 'OpenBSD']
(3, 4, 5)

mengakses semua elemen dengan looping for :
----------------------------
Zorin OSUbuntuFreeBSDNetBSDOpenBSDBacktrackFedoraSlackware

0123456789

nama : Wiro Sablengprodi : ilmu komputeremail : wirosableng@localhostwebsite :
http://www.sitampanggarang.com
(komputerteknik) D:\ECLIPSE-PYTHON\PythonBasics\Python_06_StrukturDataLanjut>
```

Mengubah Isi List, Tuple, dan Dictionary

- Ada waktunya Anda ingin mengubah salah satu elemen setelah mendefinisikan stuktur data.
- Misal ada sebuah list daftar_barang dan Anda ingin mengubah elemen ke-7 dengan data baru yang asalnya “kursi” menjadi “meja”.
- Atau ada sebuah informasi dalam bentuk dictionary dengan *key* “nama” yang *value* asalnya “Son Go Ku” menjadi “Vash De Stampede”.
- Cara mengubah data salah satu elemen di struktur data python mudah sekali. Tinggal tentukan indeks mana yang akan diubah, kemudian masukkan nilai baru kedalam indeks tersebut.

Mengubah Isi List, Tuple, dan Dictionary

Lebih jelasnya coba lihat contoh Kode Program **strukdat3.py**

```
# cara mendefinisikan list
sebuah_list = ['Zorin OS',
               'Ubuntu',
               'FreeBSD',
               'NetBSD',
               'OpenBSD',
               'Backtrack',
               'Fedora',
               'Slackware']

# cara mendefinisikan tuple
sebuah_tuple = (0, 1, 2, 3, 4, 5, 6, 7, 8, 9)

# cara mendefinisikan dictionary
sebuah_dictionary = {'nama': 'Wiro Sableng',
                     'prodi': 'ilmu komputer',
                     'email': 'wirosableng@localhost',
                     'website': 'http://www.sitampanggarang.com'
                     }
```

Mengubah Isi List, Tuple, dan Dictionary

```
# cara update sebuah elemen :
print("cara update sebuah elemen : ")
print("\n")

print(sebuah_list)
sebuah_list[5] = 'Kali Linux'
print(sebuah_list)
print("\n")

print(sebuah_tuple)
# tuple tidak dapat melakukan operasi perubahan elemen :D
#sebuah_tuple[5] = 100
print(sebuah_tuple)
print("\n")

print(sebuah_dictionary)
sebuah_dictionary['email'] = 'wiro.sableng@gmail.com'
print(sebuah_dictionary)
print("\n\n")
```

Mengubah Isi List, Tuple, dan Dictionary

Output Program **strukdat3.py**

```
cara update sebuah elemen :

['Zorin OS', 'Ubuntu', 'FreeBSD', 'NetBSD', 'OpenBSD',
'Backtrack', 'Fedora', 'Slackware']
['Zorin OS', 'Ubuntu', 'FreeBSD', 'NetBSD', 'OpenBSD', 'Kali
Linux', 'Fedora', 'Slackware']

(0, 1, 2, 3, 4, 5, 6, 7, 8, 9)
(0, 1, 2, 3, 4, 5, 6, 7, 8, 9)

{'nama': 'Wiro Sableng', 'prodi': 'ilmu komputer', 'email':
'wirosableng@localhost', 'website':
'http://www.sitampanggarang.com'}
{'nama': 'Wiro Sableng', 'prodi': 'ilmu komputer', 'email':
'wiro.sableng@gmail.com', 'website':
'http://www.sitampanggarang.com'}
```

Menambahkan Data pada List, Tuple, dan Dictionary

- Ketiga struktur data ini pun dapat ditambahkan data baru dari data semula.
- Pada list, digunakan tanda “+” untuk menambahkan data dari list baru ke list lama.
- Begitupun dengan tuple, tanda “+” digunakan untuk menambahkan data dari tuple baru ke tuple lama.
- Sedangkan pada dictionary digunakan method update dari dictionary yang ingin ditambahkan data baru. Kemudian dictionary semula akan memiliki data yang ditambahkan melalui method tersebut.
- Berikut adalah contoh menambahkan data baru pada ketiga struktur data tersebut.

Menambahkan Data pada List, Tuple, dan Dictionary

Kode Program **strukdat4.py**

```
# cara mendefinisikan list
sebuah_list = ['Zorin OS',
               'Ubuntu',
               'FreeBSD',
               'NetBSD',
               'OpenBSD',
               'Backtrack',
               'Fedora',
               'Slackware']

# cara mendefinisikan tuple
sebuah_tuple = (0, 1, 2, 3, 4, 5, 6, 7, 8, 9)

# cara mendefinisikan dictionary
sebuah_dictionary = {'nama': 'Wiro Sableng',
                     'prodi': 'ilmu komputer',
                     'email': 'wirosableng@localhost',
                     'website': 'http://www.sitampanggarang.com'
                     }
```

Menambahkan Data pada List, Tuple, dan Dictionary

```
print(sebuah_list)
list_baru = sebuah_list + ['PC Linux OS', 'Blankon', 'IGOS', 'OpenSUSE']
print(list_baru)
print("\n")

print(sebuah_tuple)
tuple_baru = sebuah_tuple + (100, 200, 300)
print(tuple_baru)
print("\n")

print(sebuah_dictionary)
dictionary_baru = {'telp': '022-12345678', 'alamat': 'Bandung, Jabar'}
sebuah_dictionary.update(dictionary_baru)
print(sebuah_dictionary)
print("\n\n")
```

Menambahkan Data pada List, Tuple, dan Dictionary

Output Program **strukdat4.py**

```
['Zorin OS', 'Ubuntu', 'FreeBSD', 'NetBSD', 'OpenBSD', 'Backtrack',
'Fedora', 'Slackware']
['Zorin OS', 'Ubuntu', 'FreeBSD', 'NetBSD', 'OpenBSD', 'Backtrack',
'Fedora', 'Slackware', 'PC Linux OS', 'Blankon', 'IGOS', 'OpenSUSE']

(0, 1, 2, 3, 4, 5, 6, 7, 8, 9)
(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 100, 200, 300)

{'nama': 'Wiro Sableng', 'prodi': 'ilmu komputer', 'email': 'wirosab
leng@localhost', 'website': 'http://www.sitampanggarang.com'}
{'nama': 'Wiro Sableng', 'prodi': 'ilmu komputer', 'email': 'wirosab
leng@localhost', 'website': 'http://www.sitampanggarang.com', 'telp'
: '022-12345678', 'alamat': 'Bandung, Jabar'}
```

Menghapus Isi List, Tuple, dan Dictionary

- Tidak lengkap rasanya bila dalam sebuah informasi ada yang tidak dapat dihapus.
- Terkadang ada sebuah data yang tidak Anda butuhkan dan ingin Anda hilangkan dari kumpulan data yang dimiliki.
- Misal dalam sebuah list Anda ingin menghapus salah satu elemen. Atau di dictionary, Anda ingin menghilangkan salah satu key dari dictionary tersebut.
- Di python sendiri penghapusan salah satu elemen dapat dilakukan di list dan dictionary. Sedangkan tuple tidak mendukung penghapusan elemen.
- Jika kita lakukan penghapusan pada salah satu elemen di tuple, maka akan muncul pesan error : “TypeError: 'tuple' object doesn't support item deletion”.
- Pada list Anda tinggal menunjuk salah satu elemennya dengan sebuah angka dari 0 sampai panjang list tersebut dikurangi satu dengan diapit tanda “[“ dan “]”.
- Sedangkan pada dictionary Anda tunjuk salah satu key yang akan dihapus dari dictionary.

Menghapus Isi List, Tuple, dan Dictionary

Berikut adalah contoh penghapusan salah satu elemen pada list dan dictionary. **Kode Program strukdat5.py**

```
# cara mendefinisikan list
sebuah_list = ['Zorin OS',
               'Ubuntu',
               'FreeBSD',
               'NetBSD',
               'OpenBSD',
               'Backtrack',
               'Fedora',
               'Slackware']

# cara mendefinisikan tuple
sebuah_tuple = (0, 1, 2, 3, 4, 5, 6, 7, 8, 9)

# cara mendefinisikan dictionary
sebuah_dictionary = {'nama': 'Wiro Sableng',
                     'prodi': 'ilmu komputer',
                     'email': 'wirosableng@localhost',
                     'website': 'http://www.sitampanggarang.com'
                     }
```

Menghapus Isi List, Tuple, dan Dictionary

```
# cara delete sebuah elemen :
print("cara delete sebuah elemen : ")
print()

print(sebuah_list)
del sebuah_list[0]
print(sebuah_list)
print()

print(sebuah_tuple)
# tuple tidak mendukung proses penghapusan elemen :D.(coba hilangkan
 tanda '#' disampingnya)
#del sebuah_tuple[8]
print(sebuah_tuple)
print()

print(sebuah_dictionary)
del sebuah_dictionary['website']
print(sebuah_dictionary)
print("\n")
```

Menghapus Isi List, Tuple, dan Dictionary

Output **Program strukdat5.py**

```
cara delete sebuah elemen :

['Zorin OS', 'Ubuntu', 'FreeBSD', 'NetBSD', 'OpenBSD', 'Backtrack',
'Fedora', 'Slackware']
['Ubuntu', 'FreeBSD', 'NetBSD', 'OpenBSD', 'Backtrack', 'Fedora', 'S
lackware']

(0, 1, 2, 3, 4, 5, 6, 7, 8, 9)
(0, 1, 2, 3, 4, 5, 6, 7, 8, 9)

{'nama': 'Wiro Sableng', 'prodi': 'ilmu komputer', 'email': 'wirosab
leng@localhost', 'website': 'http://www.sitampanggarang.com'}
{'nama': 'Wiro Sableng', 'prodi': 'ilmu komputer', 'email': 'wirosab
leng@localhost'}
```

Menghapus List, Tuple, dan Dictionary

- Pada contoh sebelumnya Anda hanya menghapus salah satu elemen.
- Lalu bagaimanakah jika ingin menghapus keseluruhan struktur data sehingga struktur data tersebut terhapus dari memory seluruhnya ?.
- Di Python dengan menggunakan perintah del pada sebuah struktur data maka struktur data tersebut akan dihapus sepenuhnya dari memory.
- Hal ini berlaku juga bagi variabel dan objek yang didefinisikan oleh programmer.
- Dengan hilangnya dari memory maka struktur data yang telah dihapus tidak dapat digunakan lagi oleh program yang Anda bangun.

Menghapus List, Tuple, dan Dictionary

Kode Program strukdat6.py

```
# cara mendefinisikan list
sebuah_list = ['Zorin OS',
               'Ubuntu',
               'FreeBSD',
               'NetBSD',
               'OpenBSD',
               'Backtrack',
               'Fedora',
               'Slackware']

# cara mendefinisikan tuple
sebuah_tuple = (0, 1, 2, 3, 4, 5, 6, 7, 8, 9)

# cara mendefinisikan dictionary
sebuah_dictionary = {'nama': 'Wiro Sableng',
                     'prodi': 'ilmu komputer',
                     'email': 'wirosableng@localhost',
                     'website': 'http://www.sitampanggarang.com'
                     }
```

Menghapus List, Tuple, dan Dictionary

```
# cara update sebuah elemen :
print("cara delete sebuah elemen : ")
print()

print(sebuah_list)
del sebuah_list
#print(sebuah_list)
print()

print(sebuah_tuple)
del sebuah_tuple
#print(sebuah_tuple)
print()

print(sebuah_dictionary)
del sebuah_dictionary
#print(sebuah_dictionary)
print("\n")
```

Menghapus List, Tuple, dan Dictionary

Output Program strukdat6.py

```
cara delete sebuah elemen :

['Zorin OS', 'Ubuntu', 'FreeBSD', 'NetBSD', 'OpenBSD', 'Backtrack',
'Fedora', 'Slackware']

(0, 1, 2, 3, 4, 5, 6, 7, 8, 9)

{'nama': 'Wiro Sableng', 'prodi': 'ilmu komputer', 'email': 'wirosab
leng@localhost', 'website': 'http://www.sitampanggarang.com'}
```

Menggunakan Built-in Function pada List, Tuple, dan Dictionary

- Apakah fitur – fitur manipulasi list, tuple, dan dictionary hanya terbatas pada hapus, tambah dan ubah ?.
- Python menyediakan beberapa fitur dasar lainnya yang dapat digunakan untuk proses mencari nilai maksimum dan minimum, menghitung panjang, membandingkan dua buah struktur data yang sejenis, bahkan mengubah struktur data dari list ke tuple atau sebaliknya.

Menggunakan Built-in Function pada List, Tuple, dan Dictionary

- Untuk mencari nilai maksimum pada list, tuple, atau dictionary digunakan function **max()**, sedangkan untuk mencari nilai minimum digunakan function **min()**.
- Untuk perbandingan dua buah struktur data sejenis, misal list dengan list, digunakanlah function **collections.Counter()** dikombinasi dengan operator perbandingan **==**. Function collections.Counter() plus == ini akan menghasilkan dua nilai yaitu True jika list pertama sama dengan list kedua, False jika kedua list tidak sama.
- Kemudian untuk mencari jumlah elemen yang berada pada struktur data tersebut digunakan function len().
- Dan terdapat juga untuk konversi tipe struktur data. Tapi fitur ini hanya dapat digunakan pada list dan tuple. Dictionary tidak mendukung proses konversi.
- Jadi hanya pengubahan dari list ke tuple dan sebaliknya. Untuk pengubahan dari list ke tuple digunakan function **tuple()** sedangkan untuk pengubahan dari tuple ke list digunakan function **list()**.

Menggunakan Built-in Function pada List, Tuple, dan Dictionary

- Built-in function lain untuk operasi pada List, Tuple, dan Dictionary adalah : reduce(), map(), sort(), set(), dlsb. Silahkan anda eksplorasi dari sumber-sumber lain.
- Agar lebih paham cobalah sedikit source code tentang penggunaan built-in function yang digunakan untuk manipulasi list, tuple, dan dictionary.

Menggunakan Built-in Function pada List, Tuple, dan Dictionary

Kode Program strukdat7.py

```
import collections

# cara mendefinisikan list
sebuah_list = ['Zorin OS',
               'Ubuntu',
               'FreeBSD',
               'NetBSD',
               'OpenBSD',
               'Backtrack',
               'Fedora',
               'Slackware']

# cara mendefinisikan tuple
sebuah_tuple = (0, 1, 2, 3, 4, 5, 6, 7, 8, 9)

# cara mendefinisikan dictionary
sebuah_dictionary = {'nama': 'Wiro Sableng',
                     'prodi': 'ilmu komputer',
                     'email': 'wirosableng@localhost',
                     'website': 'http://www.sitampanggarang.com'
                     }
```

Menggunakan Built-in Function pada List, Tuple, dan Dictionary

```
# menambahkan elemen baru
print("menambahkan elemen baru : \n")
print(sebuah_list)
list_baru = sebuah_list + ['PC Linux OS', 'Blankon', 'IGOS', 'OpenSU
SE']
print(list_baru)
print("\n")

print(sebuah_tuple)
tuple_baru = sebuah_tuple
print(tuple_baru)
print("\n")

print(sebuah_dictionary)
dictionary_baru = {'telp': '022-
12345678', 'alamat': 'Bandung, Jabar'}
sebuah_dictionary.update(dictionary_baru)
print(sebuah_dictionary)
print("\n")
```

Menggunakan Built-in Function pada List, Tuple, dan Dictionary

```
# membandingkan yang lama dengan yang baru
print("membandingkan yang lama dengan yang baru :")
print("sebuah_list banding list_baru : ")
if collections.Counter(sebuah_list) == collections.Counter(list_baru):
    print("sebuah_list adalah sama dengan list_baru")
else:
    print("sebuah_list tidak sama dengan list_baru")
print()

print("sebuah_tuple banding tuple_baru : ")
if collections.Counter(sebuah_tuple) == collections.Counter(tuple_baru):
    print("sebuah_tuple adalah sama dengan tuple_baru")
else:
    print("sebuah_tuple tidak sama dengan tuple_baru")
print()

print("sebuah_dictionary banding dictionary_baru : ")
if collections.Counter(sebuah_dictionary) == collections.Counter(dictionary_baru):
    print("sebuah_dictionary adalah sama dengan dictionary_baru")
else:
    print("sebuah_dictionary tidak sama dengan dictionary_baru")

print("\n")
```

Menggunakan Built-in Function pada List, Tuple, dan Dictionary

```
# mengetahui panjang list, tuple, dan dictionary
print("mengetahui panjang list, tuple, dan dictionary : \n")
print("panjang sebuah_list : ", len(sebuah_list))
print("panjang sebuah_tuple : ", len(sebuah_tuple))
print("panjang sebuah_dictionary : ", len(sebuah_dictionary))

print("\n")

# mengubah list, tuple, dictionary menjadi string
print("mengubah list, tuple, dictionary menjadi string : \n")
print(str(sebuah_list), ' memiliki panjang karakter : ', len(str(seb
uah_list)))
print(str(sebuah_tuple), ' memiliki panjang karakter : ', len(str(se
buah_tuple)))
print(str(sebuah_dictionary), ' memiliki panjang karakter : ', len(s
tr(sebuah_dictionary)))
```

Menggunakan Built-in Function pada List, Tuple, dan Dictionary

```
# mencari nilai max dan min
print("mencari nilai max dan min : \n")
print("coba periksa sebuah_list :")
print("max : ", max(sebuah_list))
print("min : ", min(sebuah_list))
print("\n")
print("coba periksa sebuah_tuple :")
print("max : ", max(sebuah_tuple))
print("min : ", min(sebuah_tuple))
print("\n")
print("coba periksa sebuah_dictionary :")
print("max : ", max(sebuah_dictionary))
print("min : ", min(sebuah_dictionary))
print("\n")

# mengubah list ke tuple dan sebaliknya
print("mengubah list ke tuple : ")
print("semula : ", sebuah_list)
print("menjadi : ", tuple(sebuah_list))
print("\n")
print("mengubah tuple ke list : ")
print("semula : ", sebuah_tuple)
print("menjadi : ", list(sebuah_tuple))
```

Menggunakan Built-in Function pada List, Tuple, dan Dictionary

Output Program strukdat7.py

```
menambahkan elemen baru :

['Zorin OS', 'Ubuntu', 'FreeBSD', 'NetBSD', 'OpenBSD', 'Backtrack',
'Fedora', 'Slackware']
['Zorin OS', 'Ubuntu', 'FreeBSD', 'NetBSD', 'OpenBSD', 'Backtrack',
'Fedora', 'Slackware', 'PC Linux OS', 'Blankon', 'IGOS', 'OpenSUSE']


(0, 1, 2, 3, 4, 5, 6, 7, 8, 9)
(0, 1, 2, 3, 4, 5, 6, 7, 8, 9)


{'nama': 'Wiro Sableng', 'prodi': 'ilmu komputer', 'email':
'wirosableng@localhost', 'website':
'http://www.sitampanggarang.com'}
{'nama': 'Wiro Sableng', 'prodi': 'ilmu komputer', 'email':
'wirosableng@localhost', 'website':
'http://www.sitampanggarang.com', 'telp': '022-12345678', 'alamat':
'Bandung, Jabar'}
```

Menggunakan Built-in Function pada List, Tuple, dan Dictionary

```
membandingkan yang lama dengan yang baru :
sebuah_list banding list_baru :
sebuah_list tidak sama dengan list_baru

sebuah_tuple banding tuple_baru :
sebuah_tuple adalah sama dengan tuple_baru

sebuah_dictionary banding dictionary_baru :
sebuah_dictionary tidak sama dengan dictionary_baru


mengetahui panjang list, tuple, dan dictionary :

panjang sebuah_list :  8
panjang sebuah_tuple :  10
panjang sebuah_dictionary :  6
```

Menggunakan Built-in Function pada List, Tuple, dan Dictionary

```
mengubah list, tuple, dictionary menjadi string :

['Zorin OS', 'Ubuntu', 'FreeBSD', 'NetBSD', 'OpenBSD', 'Backtrack',
'Fedora', 'Slackware']  memiliki panjang karakter :  90
(0, 1, 2, 3, 4, 5, 6, 7, 8, 9)  memiliki panjang karakter :  30
{'nama': 'Wiro Sableng', 'prodi': 'ilmu komputer', 'email':
'wirosableng@localhost', 'website':
'http://www.sitampanggarang.com', 'telp': '022-12345678', 'alamat':
'Bandung, Jabar'}  memiliki panjang karakter :  181
mencari nilai max dan min :

coba periksa sebuah_list :
max :  Zorin OS
min :  Backtrack


coba periksa sebuah_tuple :
max :  9
min :  0
```

Menggunakan Built-in Function pada List, Tuple, dan Dictionary

```
coba periksa sebuah_dictionary :
max :  website
min :  alamat


mengubah list ke tuple :
semula :  ['Zorin OS', 'Ubuntu', 'FreeBSD', 'NetBSD',
'OpenBSD', 'Backtrack', 'Fedora', 'Slackware']
menjadi :  ('Zorin OS', 'Ubuntu', 'FreeBSD', 'NetBSD',
'OpenBSD', 'Backtrack', 'Fedora', 'Slackware')


mengubah tuple ke list :
semula :  (0, 1, 2, 3, 4, 5, 6, 7, 8, 9)
menjadi :  [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Tugas

Lakukan percobaan pada VS Code untuk Program strukdat3.py, strukdat6.py, dan strukdat7.py.

Jawaban tugas ini adalah :

1. Screenshoot capture pada VScode program python yang anda ketik.
2. Screenshoot capture pada VScode hasil running program tsb pada window terminal VScode.

Hasil screenshoot bisa di rubah menjadi format pdf atau apapun sehingga sedemikian resolusi atau ukuran file total tidak melebihi 2MB seperti yang dipersyaratkan ketika mengupload jawaban anda.

Thank You

Programming for Science

Membuat Function

Pengenalan Function Tanpa “return”

- Pada saat membuat program terkadang kita menyalin blok kode yang sama di baris berikutnya, misal untuk mencetak sebuah biodata kita salin kembali kemudian ganti nilai – nilainya untuk menampilkan biodata tersebut.
- Apakah harus menyalin blok kode di setiap bagian kode yang membutuhkan kode tersebut ???. Waktu dulu kuliah Algoritma dan Pemrograman, ketika mempelajari pemrograman pertama kalinya, baris kode yang sama dipakai berulang – ulang untuk mengeluarkan hasil tertentu. Tapi jadi tidak efisien, karena ukuran *file* program yang ditulis cukup besar.
- Oleh karena itu hampir di setiap bahasa pemrograman terdapat fitur yang dinamakan ***function***. Nama lainnya *method*, *sub routine*, atau fungsi.
- *Function* ini berguna untuk penggunaan kembali blok kode yang akan digunakan di baris kode lain. Sekali tulis tinggal panggil.

Pengenalan Function Tanpa “return”

- *Function* ini jelas beda dengan *built-in function* yang ada di Python.
- *Built-in function* sendiri adalah *function* yang telah dibuatkan oleh pengembang bahasa pemrograman Python.
- Sedangkan *function* dibuat oleh *programmer* yang menggunakan bahasa pemrograman Python, atau istilah lainnya *userdefined function*.

Pengenalan Function Tanpa “return”

- Di Python untuk membuat *function* digunakan ***keyword def***. Kemudian diikuti nama *function* yang diinginkan lalu parameter yang dibutuhkan dengan diawali tanda “(“ dan “)”.
- Untuk membuka *function* dimulai dengan tanda “:”. Tidak seperti di C, Java, atau PHP yang diawali dengan tanda “{“ dan diakhiri “}” untuk membuka sebuah function.
- Lalu tutupnya ? Seperti dijelaskan diawal di Python sendiri digunakan indentasi untuk menentukan apakah baris kode tersebut milik sebuah *function*, *if*, atau pengulangan.
- Jadi jika Anda ingin menuliskan kode untuk *function* yang Anda buat. Harus ada jarak satu indentasi agar kode tersebut dianggap sebagai kode dari *function* yang Anda tulis.
- Kemudian Anda dapat menambahkan *keyword* ***return*** jika *function* yang Anda tulis ingin mengembalikan nilai keluaran

Pengenalan Function Tanpa “return”

Berikut ada contoh tentang pembuatan *function* yang memiliki parameter dan tidak memiliki parameter. Penggunaan *return* digunakan pada contoh berikutnya :

Kode program *fungsi1.py*

```
def fungsi_tanpa_parameter():
   for i in range(1, 5):
      print("looping ke - ", i)

def fungsi_berparameter(batas_akhir):
   for i in range(1, batas_akhir):
      print("looping ke - ", i)

print("contoh penggunaan fungsi tanpa parameter : ")
print("hasil : ", fungsi_tanpa_parameter())
print9"\n\n")

print("contoh penggunaan fungsi berparameter : ")
print("hasil : ", fungsi_berparameter(10))
```

Pengenalan Function Tanpa “return”

- Sebuah *function* jika dipanggil langsung nilai keluarannya tidak akan dicetak.
- Tetapi jika dipanggil melalui sebuah *function* seperti **print** nilai keluarannya akan ditampilkan.
- Kenapa nilainya “None” ? Karena di *function* yang tadi ditulis tidak disertakan *keyword* **return**.
- Jika sebuah *function* tidak diberikan **return** maka dapat dibilang *function* tersebut dianggap *procedure*. Sebuah *function* yang tidak memiliki nilai keluaran.

Pengenalan Function Tanpa “return”

Output program *fungsi1.py*

```
contoh penggunaan fungsi tanpa parameter :
looping ke -  1
looping ke -  2
looping ke -  3
looping ke -  4
hasil :  None


contoh penggunaan fungsi berparameter :
looping ke -  1
looping ke -  2
looping ke -  3
looping ke -  4
looping ke -  5
looping ke -  6
looping ke -  7
looping ke -  8
looping ke -  9
hasil :  None
```

Function yang Menggunakan “return”

- Bagaimana kalau ditambahkan **return** ?
- Jika ditambahkan **return**, *function* yang Anda tulis akan menghasilkan nilai keluaran.
- Jika dipanggil langsung maka program tidak akan menampilkan nilai keluaran dari *function* tersebut.
- Jika *function* tersebut dipanggil melalui *function* atau *keyword* misalnya **print**, maka nilai keluarannya akan ditampilkan.
- Berikut terdapat *function* yang menghasilkan nilai keluaran yang memiliki parameter dan tidak berparameter :

Function yang Menggunakan "return"

Kode program *fungsi2.py*

```
def fungsi_tanpa_parameter():
    temp = 0
    for i in range(1, 5):
        temp = temp + i
    return temp


def fungsi_berparameter(batas_akhir):
    temp = 0
    for i in range(1, batas_akhir):
        temp = temp + i
    return temp


print("contoh penggunaan fungsi tanpa parameter : ")
print("hasil : ", fungsi_tanpa_parameter())
print("hasil : ", fungsi_tanpa_parameter())
print("hasil : ", fungsi_tanpa_parameter())
print("\n")
print("contoh penggunaan fungsi berparameter : ")
print("hasil : ", fungsi_berparameter(15))
print("hasil : ", fungsi_berparameter(20))
print("hasil : ", fungsi_berparameter(25))
```

Function yang Menggunakan “return”

- Anda sendiri dapat melihat perbedaannya antara *function* yang berparameter dengan tidak berparameter.
- Pada *function* yang tidak berparameter. Ketika dipanggil berulang – ulang nilai keluarannya tetap sama.
- Berbeda dengan *function* yang memiliki parameter, nilai keluaranya berbeda – beda ketika dipanggil. Tergantung nilai masukan yang diberikan.

Function yang Menggunakan “return”

Output program *fungsi2.py*

```
contoh penggunaan fungsi tanpa parameter :
hasil :  10
hasil :  10
hasil :  10


contoh penggunaan fungsi berparameter :
hasil :  105
hasil :  190
hasil :  300
```

Default Argument pada Python

- Sekarang Anda sudah mengenal *function* yang berparameter dan tidak berparameter.
- Umumnya saat akan memberikan nilai pada sebuah function, nilai tersebut akan diberikan saat function tersebut dipanggil. Apakah saat memasukkan nilai boleh tidak diisi atau dilewat ? Lalu apakah akan ada nilainya ?.
- Di Python terdapat sebuah fitur yang dinamakan default argument saat menulis sebuah *function*.
- Default argument sendiri adalah sebuah argumen yang sudah diisi nilai terlebih dahulu jika argumen tersebut tidak diberikan saat memanggil *function*.
- Jadi sekalipun dilewat nilai dari argument tersebut akan dipenuhi dengan nilai *default* nya.

Default Argument pada Python

- Berikut dibawah ini terdapat contoh pemanggilan *function* yang melewatkan semua argumen yang dibutuhkan *function*, dan yang tidak melewatkan semua argumen yang akan ditangani oleh *default argument* :

Kode program *fungsi3.py*

```
def cetak_biodata(nama, kota, umur=18):
    print("Nama : ", nama)
    print("Umur : ", umur)
    print("Kota : ", kota)
    return


# kalau parameter diisi semua
print("Tanpa memakai default argument : ")
cetak_biodata(nama="miki", umur=50, kota="bandung")
print("\n")

# kalau parameter tidak diisi semua
print("Memakai default argument : ")
cetak_biodata(kota="bandung", nama="miki")
```

Default Argument pada Python

Output program *fungsi3.py*

```
Tanpa memakai default argument :
Nama :  miki
Umur :  50
Kota :  bandung


Memakai default argument :
Nama :  miki
Umur :  18
Kota :  bandung
```

Variable-length Argument pada Python

- Sekarang kita akan mengenal fitur yang dinamakan dengan *variable-length argument*.
- Fitur ini digunakan ketika ingin membuat sebuah *function* yang memiliki argumen yang dinamis.
- Argumen ini dapat disebut sebagai argumen yang tidak utama. Misal dalam sebuah fungsi dibutuhkan tiga argumen, maka argumen ke – 4 sampai ke – n argumen, tidak akan ditampung oleh argumen utama.
- Tapi ditampung oleh argumen terakhir yang menampung seluruh argumen yang diberikan setelah argumen utama.
- Di Python untuk menandakan bahwa argumen tersebut *variable-length argument*, diberikan tanda "*" pada argumen terakhir. *Variable-length argument* ini harus disimpan di akhir setelah argumen biasa dan *default argument*. Apabila disimpan di urutan awal, maka Python akan mengeluarkan error : "SyntaxError: invalid syntax".

Variable-length Argument pada Python

- Sebagai contoh *Variable-length Argument* pada Python dapat diperhatikan kode berikut ini.

Kode program *fungsi4.py*

```
def cetak_perolehan_nilai(nama, twitter, *scores):
    print("Nama : ", nama)
    print("Twitter : ", twitter)
    print("Score yang diperoleh : ")
    i = 1
    for score in scores:
        print("nilai ke - %d : %d" % (i, score))
        i = i + 1
    return

# kalau parameter diisi semua
print("Dengan adanya variable-length variabel sisa akan menjadi
tuple : ")
cetak_perolehan_nilai("Silvy", "@sivlysiv", 90, 80, 70, 80, 90)
```

Variable-length Argument pada Python

- Seperti yang Anda lihat pada contoh diatas, argumen utama adalah nama dan twitter.
- Apabila kita memasukkan argumen setelahnya, maka argumen tersebut akan dikumpulkan dalam satu wadah yaitu *scores. Berapapun kita masukkan argumen, akan ditampung menjadi sebuah list yang berisi argumen – argumen yang dimasukkan setelah nama dan twitter.

Output program *fungsi4.py*

```
Dengan adanya variable-length variabel
sisa akan menjadi tuple :
Nama :  Silvy
Twitter :  @sivlysiv
Score yang diperoleh :
nilai ke - 1 : 90
nilai ke - 2 : 80
nilai ke - 3 : 70
nilai ke - 4 : 80
nilai ke - 5 : 90
```

Keyword Argument pada Function

- Dalam penggunaan *function* Anda mesti melewatkan argumen sesuai urutan yang ditulis pada parameter yang dibutuhkan oleh *function* yang Anda tulis.
- Apakah mungkin jika ditulis tanpa urutan argumen sudah baku pada *function* tersebut.
- Dalam *function* terdapat sebuah fitur yang dinamakan *keyword argument*.
- *Keyword argument* ini dapat melewatkan argumen tanpa harus sesuai urutan. *Keyword argument* diberikan saat memanggil sebuah *function* dengan mengambil nama argumen yang terdapat di *function* disambung dengan tanda “=” dan nilai dari argumen tersebut.
- Jika kita memberikan argumen yang tidak sesuai urutan tanpa menggunakan *keyword argument*, maka argumen yang diterima *function* tersebut tidak akan sesuai.

Keyword Argument pada Function

Kode Program *fungsi5.py*

```
def cetak_biodata(nama, umur, kota):
    print("Nama : ", nama)
    print("Umur : ", umur)
    print("Kota : ", kota)
    return

# kalau pakai keyword argument : mau urutannya gimanapun input akan sesuai
print("Tanpa memakai keyword argument : ")
cetak_biodata(kota="bandung", nama="miki", umur=50)
print("\n")

# kalau tidak memakai keyword argument : mau urutannya gimanapun input tidak a
kan sesuai
print("Memakai keyword argument : ")
cetak_biodata("bandung", "miki", 50)
print("\n")

# kalau tidak memakai keyword argument : tapi urutannya sesuai maka input akan
 sesuai
print("Memakai keyword argument : tapi urutannya sesuai ")
cetak_biodata("miki", 50, "bandung")
```

Keyword Argument pada Function

- Pada contoh diatas, Anda dapat melihat perbedaan antara *function* yang melewatkan *keyword argument* dengan yang tidak menggunakan *keyword argument*.
- Contoh yang tidak menggunakan *keyword argument* tidak akan menerima masukan sesuai yang dibutuhkan *function* ketika urutan argumennya diacak.

Output Program *fungsi5.py*

```
Tanpa memakai keyword argument :
Nama :  miki
Umur :  50
Kota :  bandung

Memakai keyword argument :
Nama :  bandung
Umur :  miki
Kota :  50

Memakai keyword argument : tapi urutannya sesuai
Nama :  miki
Umur :  50
Kota :  bandung
```

Keyword-length Argument pada Function

- *Keyword-length argument* mempunyai cara penggunaan yang sama hanya saja, *keyword-length* ini menampung *keyword argument* yang berlebih ketika diberikan kepada *function* yang dipanggil.
- *Keyword argument* yang berlebih akan diterima dalam bentuk **dictionary**.

Keyword-length Argument pada Function

Kode Program *fungsi6.py*

```
def cetak_perolehan_nilai(nama, twitter, **data_tambahan):
    print("Nama : ", nama)
    print("Twitter : ", twitter)
    print()
    print("Data Lainnya : ")
    i = 1
    for data in data_tambahan:
        print("%s : %s" % (data, data_tambahan[data]))
    return


# kalau parameter diisi semua
print("Dengan adanya keyword argument, argumen tersisa akan m
enjadi dictionary : ")
cetak_perolehan_nilai("Silvy", "@sivlysiv", email="silvysilvy
@gmail.com", facebook="www.facebook.com/silvysil", telp="0838
-1234-5678")
```

Keyword-length Argument pada Function

- Pada contoh diatas, *keyword argument* yang berlebih ditampung kedalam *argument* data_tambahan dan argumen berlebih tersebut disimpan dalam **dictionary**.

Output Program *fungsi7.py*

```
Dengan adanya keyword argument, argumen tersisa akan menjadi dictionary :
Nama :  Silvy
Twitter :  @sivlysiv

Data Lainnya :
email : silvysilvy@gmail.com
facebook : www.facebook.com/silvysil
telp : 0838-1234-5678
```

Pass by Reference dan Pass by Value pada Python

- Berikutnya terdapat masalah *pass by value* atau *pass by reference*.
- Di Python semua nilai akan dilewatkan secara *by reference*. Artinya jika kita mengubah argumen di dalam fungsi maka nilai argumen yang direferensi tersebut akan ikut berubah juga.
- Misalkan dibawah contoh berikut terdapat sebuah **list** yang akan diganti dengan nilai baru, dan ada juga yang ditambahkan nilai baru.

Pass by Reference dan Pass by Value pada Python

Kode Program
fungsi7.py

```
def sebuah_fungsi(sebuah_list):
    sebuah_list = [1, 2, 3, 4, 5]
    print(sebuah_list)

def sebuah_fungsi_lainnya(sebuah_list):
    sebuah_list.append([10, 20, 30])
    print(sebuah_list)

ini_list = [10, 20, 30]
sebuah_list = [100, 200, 300]

print("apakah ini_list berubah ? ")
print(ini_list)
sebuah_fungsi(ini_list)
print(ini_list)
print(ini_list)
sebuah_fungsi_lainnya(ini_list)
print(ini_list)

print("apakah sebuah_list berubah ? ")
print(sebuah_list)
sebuah_fungsi(sebuah_list)
print(sebuah_list)
print(sebuah_list)
sebuah_fungsi_lainnya(sebuah_list)
print(sebuah_list)
```

Pass by Reference dan Pass by Value pada Python

- Pada kode diatas, Anda akan melihat sebuah perbedaan yang cukup penting.
- Ketika sebuah **list** diganti nilainya maka **list** yang ada di luar *function* tidak akan terpengaruh.
- Tapi ketika kita menambahkan data baru dengan menggunakan *method* pada **list** tersebut. Nilai diluar ikut berubah,. Hal ini terjadi karena pada *function* sebuah_fungsi_lainnya(), **list** sebuah_list masih menunjuk atau merujuk ke sebuah_list yang berada diluar. Atau dalam kata lain masih menunjuk ke “address” yang sama di memori utama.

Pass by Reference dan Pass by Value pada Python

Output Program
fungsi7.py

```
apakah ini_list berubah ?
[10, 20, 30]
[1, 2, 3, 4, 5]
[10, 20, 30]
[10, 20, 30]
[10, 20, 30, [10, 20, 30]]
[10, 20, 30, [10, 20, 30]]
apakah sebuah_list berubah ?
[100, 200, 300]
[1, 2, 3, 4, 5]
[100, 200, 300]
[100, 200, 300]
[100, 200, 300, [10, 20, 30]]
[100, 200, 300, [10, 20, 30]]
```

Variable Scope pada Python

- *Variable scope* adalah sebuah kondisi dimana variabel diakses secara lokal pada blok kode tertentu atau bersifat universal yang menyebabkan variabel tersebut dapat diakses dari blok kode manapun.
- Misal ada sebuah variabel di dalam *function*. Variabel tersebut bersifat lokal dan hanya dapat diakses didalam *function* tersebut.
- Lalu bagaimanakah kita menjadikan sebuah variabel agar bersifat global ?. Di Python terdapat sebuah *keyword* yang bernama **global**. *Keyword* ini digunakan untuk merujuk sebuah variabel di luar blok kode, misalnya sebuah variabel di dalam *function*, dengan nama yang sama.

Variable Scope pada Python

Kode Program *fungsi8.py*

```
def sebuah_fungsi():
    angka = 10
    print("di dalam sebuah_fungsi, angka bernilai : ", angka)

def sebuah_fungsi_lainnya():
    global angka
    angka = 114
    print("di dalam sebuah_fungsi, angka bernilai : ", angka)

angka = 6666

print("sebelum dipanggil sebuah_fungsi : ", angka)
sebuah_fungsi()
print("sesudah dipanggil sebuah_fungsi : ", angka)

print()

print("sebelum dipanggil sebuah_fungsi_lainnya : ", angka)
sebuah_fungsi_lainnya()
print("sesudah dipanggil sebuah_fungsi_lainnya : ", angka)
```

Variable Scope pada Python

- Pada kode diatas variabel yang bernama angka dibubuhi *keyword* **global** pada *function* sebuah_fungsi_lainnya().
- Hasilnya saat angka diubah nilainya. Maka nilai di variabel angka yang berada di luar blok *function* sebuah_fungsi_lainnya() ikut berubah.

Output Program *fungsi8.py*

```
sebelum dipanggil sebuah_fungsi :  6666
di dalam sebuah_fungsi, angka bernilai :  10
sesudah dipanggil sebuah_fungsi :  6666

sebelum dipanggil sebuah_fungsi_lainnya :  6666
di dalam sebuah_fungsi, angka bernilai :  114
sesudah dipanggil sebuah_fungsi_lainnya :  114
```

Thank You

Programming for Science

Mengenal Exception Dan With statement

Jenis-Jenis Exception

- *Exception* adalah sebuah cara di Python untuk menjebak *error*, dan menangani *error* tak terduga pada program Python yang Anda tulis.
- *Exception* akan tetap menjalankan baris kode program dibawah bagian kode program yang *error*. Hal ini mempermudah proses *debugging*.
- Lalu apa bedanya jika kita menggunakan kondisional biasa yang menggunakan **if** untuk mencegah *error* ?
- Pertama Anda harus mencari cara untuk menangkap nilai – nilai yang *error*, misal ketika membuka *file* Anda harus menggunakan method – method yang ada pada *file* untuk mengetahui *error* atau tidak.

Jenis-Jenis Exception

- Kedua dengan menggunakan kondisional **if** biasa, program yang Anda tulis akan langsung dihentikan ketika *error* terjadi.
- Ketiga pengambilan *error* akan otomatis ditangani oleh Python dan *error* tersebut akan ditangani sesuai dengan penanganan yang Anda lakukan, dan baris kode program dibawahnya akan tetap dieksekusi.
- Python sendiri sudah menyediakan beberapa standard *error* yang dapat digunakan oleh programmer dalam menjaga pengembangan aplikasinya dari *error* yang tak terduga.
- Anda sendiri dapat membuat error menurut definisi Anda. Hal tersebut akan diulas di bagian akhir bab ini.
- Berikut adalah beberapa standard *error* yangt terdapat di Python :

Jenis-Jenis Exception

No	Nama Exception	Keterangan
1	Exception	Menangani semua *exception*
2	StopIteration	*Exception* ini muncul ketika *method* next() tidak menunjuk ke objek apapun saat iterasi
3	SystemExit	*Exception* ini muncul ketika sys.exit() dipanggil
4	StandardError	*Exception* untuk menangani semua *built-in exception* kecuali StopIteration dan SystemExit
5	ArithmeticError	*Exception* untuk menangani *error* saat perhitungan angka
6	OverflowError	*Exception* ini muncul ketika perhitungan angka melebihi batas maksimum dari tipe angka yang dihitung
7	FloatingPointError	*Exception* ini muncul ketika terdapat kegagalan pada perhitungan angka bertipe *float*
8	ZeroDivisionError	*Exception* ini muncul jika ada pembagian atau modulus oleh 0 terhadap angka tipe apapun
9	AssertionError	*Exception* ini muncul ketika terjadi kegagalan pada saat perintah **assert** dijalankan

Jenis-Jenis Exception

No	Nama Exception	Keterangan
10	AssertionError	*Exception* ini muncul ketika terjadi kegagalan pada saat perintah **assert** dijalankan
11	AttributeError	*Exception* ini muncul ketika gagal menunjuk atribut dari suatu objek
12	EOFError	*Exception* ini muncul ketika tidak ada input saat menggunakan *function* raw_input() atau input dan telah mencapai bagian akhir file saat pembacaan file.
13	ImportError	*Exception* ini muncul ketika gagal saat menggunakan **import**
14	KeyboardInterrupt	*Exception* ini muncul ketika user meng-*interrupt* eksekusi program, biasanya ketika menekan kombinasi **ctrl + c**
15	LookupError	*Exception* muncul ketika gagal pada saat proses *lookup*
16	IndexError	*Exception* ini muncul ketika tidak ada indeks yang dituju pada struktur data seperti **list** atau **tuple**
17	KeyError	*Exception* ini muncul ketika tidak ada *key* yang dituju pada *dictionary*

Jenis-Jenis Exception

No	Nama Exception	Keterangan
18	NameError	*Exception* ini muncul ketika variabel tidak ditemukan pada lingkup lokal di suatu function dan kondisional atau pada lingkup global
19	UnboundLocalError	*Exception* ini muncul ketika mencoba mengakses variabel lokal di *function* atau *method* tapi belum ada nilainya
20	EnvironmentError	*Exception* in muncul ketika terjadi kegagalan diluar lingkup Python
21	IOError	*Exception* in muncul ketika proses *input/output* gagal, misal saat menggunakan **print** atau saat membuka *file*
22	OSError	*Exception* ini muncul ketika terjadi kegagalan pada sistem operasi yang digunakan
23	SyntaxError	*Exception* ini muncul ketika terjadi kesalahan pada penggunaan sintaks Python
24	IndentationError	*Exception* ini muncul ketika indentasi pada blok kode tidak sesuai penggunaannya
25	SystemError	*Exception* ini muncul ketika terdapat masalah internal pada *interpreter*, saat *error* ini muncul interpreter tidak akan keluar

Jenis-Jenis Exception

No	Nama Exception	Keterangan
26	TypeError	*Exception* ini muncul jika ada kesalahan tipe data saat proses perhitungan, misal huruf dibagi angka
27	ValueError	*Exception* ini muncul ketika argumen yang tidak sesuai diterima oleh *builtin function*
28	RuntimeError	*Exception* ini muncul ketika terjadi kesalahan yang tidak masuk kategori manapun
29	NotImplementedEror	*Exception* ini muncul ketika *abstract method* dari suatu *class* tidak diimplementasikan di *class* yang mewarisinya.

- Agar lebih paham dibawah ini ada beberapa contoh kode yang penggunaan *exception*-nya sangat sering digunakan.
- Sebagai contoh pertama berikut terdapat kode yang berisi pembagian oleh angka nol.

Jenis-Jenis Exception

Kode Program *exception1.py*

```
sebuah_angka = 29

try:
    print(sebuah_angka / 0)
except:
    print("proses perhitungan gagal ")

print("proses cetak ini masih dapat dijalankan ")

try:
    print(sebuah_angka / 0)
except ZeroDivisionError as e:
    print("proses perhitungan gagal karena : ", e)

print("proses cetak ini masih dapat dijalankan ")

print(sebuah_angka / 0)
# jika tidak memakai exception maka proses berikutnya tidak akan dijalan
kan
print("apakah proses cetak ini masih dapat dijalankan ??? ")
```

Jenis-Jenis Exception

- Struktur program adalah **`try … excep …`** .Di dalam **try** terdapat kode yang kemungkinan akan memunculkan **exception**.
- Sedangkan di dalam except adalah kode yang akan dieksekusi jika exception tersebut muncul. Pada try-except yang pertama, semua error akan ditangani dan Anda tidak akan mengetahui jenis exception apa yang yang ditangani.
- Pada try-except yang kedua, Anda memprediksi akan menangani error jika terjadi pembagian oleh nol. Manakah yang lebih baik ? Pastikan Anda sudah memiliki perkiraan apa saja error yang akan terjadi sehingga saat debugging nanti akan mempermudah Anda untuk memperbaiki kode program Anda.
- Pada blok kode try-except sekalipun error kode dibawahnya akan tetap dieksekusi. Pada proses perhitungan di bagian akhir tidak ditangani oleh try-except sehingga kode dibawahnya tidak dieksekusi.

Jenis-Jenis Exception

Output Program *exception1.py*

```
proses perhitungan gagal
proses cetak ini masih dapat dijalankan
proses perhitungan gagal karena :  division by zero
proses cetak ini masih dapat dijalankan
Traceback (most recent call last):
  File "d:/ECLIPSE-
PYTHON/PythonBasics/Python_08_MengenalException/exception1.p
y", line 18, in <module>
    print(sebuah_angka / 0)
ZeroDivisionError: division by zero
```

Jenis-Jenis Exception

- Contoh lain yang umum dipakai adalah IndexError dan KeyError.
- Kedua *error* ini umum dipakai saat operasi **list**, **tuple**, dan **dictionary**.
- Berikut terdapat contoh menunjuk indeks dan *key* yang tidak terdapat pada **list, tuple,** dan **dictionary** yang didefinisikan dalam kode dibawah ini.

Jenis-Jenis Exception

Kode Program *exception2.py*

```
sebuah_list = [1, 2, 3, 4, 5]
sebuah_tuple = (1, 2, 3, 4, 5)
sebuah_dictionary = {'nama': 'Mangaraja', 'email': 'mangaraja@yahoo.com'}

try:
    print(sebuah_list[10])
except IndexError as e:
    print("proses gagal karena : ", e)

print("proses cetak ini masih dapat dijalankan ")

try:
    print(sebuah_tuple[10])
except IndexError as e:
    print("proses gagal karena : ", e)

print("proses cetak ini masih dapat dijalankan ")

try:
    print(sebuah_dictionary['website'])
except KeyError as e:
    print("proses gagal karena : ", e)

print("proses cetak ini masih dapat dijalankan ")
```

Jenis-Jenis Exception

- Pada contoh diatas “sebuah_list” dan “sebuah_tuple” ditangani oleh *try-except* yang menangani *exception* IndexError.
- Pada masing – masing blok, kita ingin mencoba indeks yang tidak ada pada **list** dan **tuple** tersebut.
- Sedangkan pada blok *try-except* untuk *dictionary*, kita ingin mencoba menunjuk *key* “website” tapi karena key tersebut tidak ada, maka akan muncul exception KeyError.

Jenis-Jenis Exception

Output Program *exception2.py*

```
proses gagal karena :  list index out of range
proses cetak ini masih dapat dijalankan
proses gagal karena :  tuple index out of range
proses cetak ini masih dapat dijalankan
proses gagal karena :  'website'
proses cetak ini masih dapat dijalankan
```

Jenis-Jenis Exception

- Berikutnya contoh *exception* yang tak kalah populer lainnya adalah AttributeError.
- *Exception* ini muncul ketika sebuah *class* tidak memiliki atribut (variabel) yang diakses oleh *programmer*.
- Hal ini sangat penting untuk diperhatikan ketika merancang sebuah aplikasi berbasis objek. Anda harus memeriksa apakah atribut yang Anda akses pada sebuah kelas ada pada saat perancangan atau tidak.
- Jika tidak yakin gunakanlah *try-except* untuk menjebak AttributeError tersebut.

Jenis-Jenis Exception

Kode Program *exception3.py*

```
class PersegiPanjang:
    panjang = 0
    lebar = 0

    def __init__(self, p, l):
        self.panjang = p
        self.lebar = l


prsg_pjg = PersegiPanjang(10, 5)

try:
    print("panjang : ", prsg_pjg.panjang)
    print("lebar : ", prsg_pjg.lebar)
    print("tinggi :", prsg_pjg.tinggi)
except AttributeError as e:
    print("proses pemanggilan atribut gagal karena --> ", e)

print("proses cetak ini masih dapat dijalankan")
```

Jenis-Jenis Exception

- Pada contoh diatas, kita ingin mencoba mengakses atribut tinggi pada objek prsg_pjg.
- Sebelumnya tahapan yang dilalui adalah proses instansiasi, dimana kita memanggil sebuah template objek yang akan dibentuk kedalam sebuah variabel.
- Kemudian di bagian *try-except* tersebut kita coba akses atribut tinggi. Karena atribut tersebut tidak ada di kelas persegi panjang, maka *exception* AttributeError akan muncul.

Jenis-Jenis Exception

Output Program *exception3.py*

```
panjang :  10
lebar :  5
proses pemanggilan atribut gagal karena -->  'PersegiPanjang' object has no
attribute 'tinggi'
proses cetak ini masih dapat dijalankan

(komputerteknik) D:\ECLIPSE-
PYTHON\PythonBasics\Python_08_MengenalException>
```

Jenis-Jenis Exception

- Contoh yang terakhir dari sekian banyak *exception* yang terdapat di Python adalah IOError.
- *Exception* ini biasa terjadi ketika proses *input* data, saat mencetak data, atau saat operasi *file*.
- Pada contoh berikut kita akan membuka sebuah *file*, tapi *file* tersebut tidak ada.
- Secara *default* jika kita membuka *file* tanpa menyertakan mode pembacaan, maka mode tersebut adalah mode 'r' yang artinya *read* atau baca.

Jenis-Jenis Exception

Kode Program *exception4.py*

```
try:
    f = open('nilai.txt')
except IOError as e:
    print("Proses pembukaan file gagal karena : ", e)

print("proses cetak pada baris ini masih dapat dijalankan")
```

Jenis-Jenis Exception

- Pada contoh diatas kita ingin membuka *file* nilai.txt, tapi karena *file* tersebut belum pernah ditulis sebelumnya maka *exception* akan muncul yaitu IOError.
- Selain digunakan untuk *file*, IOError dapat terjadi juga saat pembacaan *built-in storage* milik Python seperti saat pembacaan pickle, shelve, dan marshal.

Jenis-Jenis Exception

Output Program *exception4.py*

```
Proses pembukaan file gagal karena :  [Errno 2] No such file or directory: 'nilai.txt'
proses cetak pada baris ini masih dapat dijalankan
```

Menyusun Multiple Except

- Apakah kita hanya dapat menangkap satu *exception* dalam satu *blok try-except* ?.
- Di Python sendiri terdapat fitur **multiple except**, yaitu kita dapat menangkap *exception* dengan baris *except* yang berbeda.
- Hal ini dilakukan jika kita ingin memberikan perlakuan berbeda kepada setiap *exception* yang ditangani.
- Lebih lemgkapnya pantau kode dibawah ini.

Menyusun Multiple Except

Kode Program *exception5.py*

```
try:
    angka1 = int(input('masukkan angka ke-1 : '))
    angka2 = int(input('masukkan angka ke-2 : '))
    print('hasil perhitungan : ', angka1 / angka2)
except ZeroDivisionError as e:
    print("proses pembagian gagal karena : ", e)
except ValueError as e:
    print("proses perhitungan gagal karena : ", e)

print("proses cetak ini masih dapat dijalankan ")
```

Menyusun Multiple Except

- Pada kode diatas kita mencoba menjebak dua buah *exception* dengan dua baris *except* berbeda.
- Hal tersebut dilakukan agar perlakuan pada penanganan setiap *exception* memiliki penanganan yang berbeda.
- Misal pada baris *except* pembagian nol ada informasi “proses pembagian gagal karena : “, sedangkan di baris *except* nilai *error* terdapat informasi “proses perhitungan gagal karena : “.
- Jadi dengan menggunakan baris *except* yang berbeda Anda dapat menangani *error* yang berbeda sesuai kebutuhan Anda.

Menyusun Multiple Except

Output Program *exception5.py*

```
masukkan angka ke-1 : 10
masukkan angka ke-2 : 0
proses pembagian gagal karena :  division by zero
proses cetak ini masih dapat dijalankan
```

```
masukkan angka ke-1 : 10
masukkan angka ke-2 : a
proses perhitungan gagal karena :  invalid literal for
int() with base 10: 'a'
proses cetak ini masih dapat dijalankan
```

Menggunakan Multiple Except

- Berbeda sedikit pada contoh sebelumnya, jika pada setiap *exception* ditangani oleh setiap baris **except**.
- Maka pada kaidah *multiple exception* di satu **except** menangani beberapa *exception*.
- Bedanya, semua *exception* yang ditangani baris **except** tersebut akan mendapat penanganan yang sama.

Menggunakan Multiple Except

Kode Program *exception6.py*

```
try:
    angka1 = float(input('masukkan angka ke-1 : '))
    angka2 = float(input('masukkan angka ke-2 : '))

    print('hasil perhitungan : ', angka1 / angka2)

except (ZeroDivisionError, ValueError, TypeError) as
e:
    print("proses perhitungan gagal karena : ", e)

print("proses cetak ini masih dapat dijalankan ")
```

Menggunakan Multiple Except

Kode *exception6.py* jika dieksekusi akan muncul tampilan seperti berikut :

```
masukkan angka ke-1 : 10
masukkan angka ke-2 : 0
proses perhitungan gagal karena :  float division by zero
proses cetak ini masih dapat dijalankan
```

```
masukkan angka ke-1 : 10
masukkan angka ke-2 : a
proses perhitungan gagal karena :  could not convert
string to float: 'a'
proses cetak ini masih dapat dijalankan
```

Try-Except Bersarang

- Mirip seperti kondisional atau perulangan yang dapat ditambahkan blok kode kondisional atau perulangan didalamnya.
- *Try-except* pun mempunyai kaidah yang sama dimana *try-except* dapat disimpan didalam *try-except* yang lainnya.
- Prioritasnya adalah tangani yang luar terlebih dahulu. Jika terjadi di *try-except* terluar maka blok kode didalamnya yang terdapat *try-except* tidak akan dieksekusi. Jika di blok luar tidak terdapat *error*. Maka penanganan *exception* di *try-except* bagian dalam akan dilakukan.

Try-Except Bersarang

Kode Program *exception7.py*

```
try:
    angka1 = float(input('masukkan angka ke-1 : '))
    angka2 = float(input('masukkan angka ke-2 : '))

    try:
        print('hasil perhitungan : ', angka1 / angka2)
    except ZeroDivisionError as e:
        print("proses perhitungan gagal karena : ", e)

except ValueError as e:
    print("proses input gagal karena : ", e)

print("proses cetak ini masih dapat dijalankan ")
```

Try-Except Bersarang

Jika pada contoh *exception5.py* baris **except** ZeroDivisionError disimpan di tingkat pertama, sekarang di *exception7.py* baris tersebut disarangkan di try-except yang utama. Dengan demikian Anda dapat menangani exception dari dalam secara langsung.

```
masukkan angka ke-1 : 10
masukkan angka ke-2 : 0
proses perhitungan gagal karena :  float division by zero
proses cetak ini masih dapat dijalankan
```

```
masukkan angka ke-1 : 10
masukkan angka ke-2 : a
proses input gagal karena :  could not convert string to
float: 'a'
proses cetak ini masih dapat dijalankan
```

Membuat Exception Sendiri

- Lalu apakah kita terbatas pada penanganan standard *exception* Python saja ?.
- Anda dapat membuat *exception* Anda sendiri dengan membuat sebuah kelas yang diturunkan dari kelas Exception.
- Dengan cara tersebut, Anda dapat membuat *exception* Anda sesuai kebutuhan pada kasus yang akan Anda tangani.
- Misal kita ingin membuat sebuah *exception* jika angka yang dimasukkan adalah angka negatif. Pertama kita buat dulu **class** nya dengan nama yang diingikan, turunkan dari kelas Exception, dan tentukan penanganan *error* pada *method – method* di kelas tersebut.

Membuat Exception Sendiri

Kode Program *exception8.py*

```
class NegativeValueError(Exception):
    def __init__(self, value):
        self.value = value

    def __str__(self):
        s = "Tidak menerima angka kurang dari 0 " + str(self.value)
        return s


def cekAngka(angka):
    if angka < 0:
        raise NegativeValueError(angka)


try:
    sebuah_angka = int(input("masukkan sebuah angka : "))
    cekAngka(sebuah_angka)
except (NegativeValueError, TypeError) as e:
    print("proses gagal karena : ", e)
```

Membuat Exception Sendiri

Untuk memanggil *exception*-nya kita memerlukan *keyword* **raise** ketika *exception* tersebut dimunculkan maka *exception* akan ditangani **except** dan mengeluarkan pesan *error*-nya. Pesan tersebut berasal dari *function* __str__() yang sebelumnya telah kita definisikan pada kelas NegativeValueError.

```
masukkan sebuah angka : 10

(komputerteknik) D:\ECLIPSE-
PYTHON\PythonBasics\Python_08_MengenalException>
```

```
masukkan sebuah angka : -100
proses gagal karena :  Tidak menerima angka kurang dari 0 -100

(komputerteknik) D:\ECLIPSE-
PYTHON\PythonBasics\Python_08_MengenalException>
```

Menggunakan “finally” pada Try-Except

- Dan akhirnya sekarang kita membahas ***finally***. *Keyword* ini digunakan untuk menentukan penanganan apa yang harus dilakukan baik ketika *exception* muncul atau tidak.
- Misal saat mengambil data dari database, kita tidak akan tahu ada kegagalan apa yang akan terjadi. Agar program kita tetap aman dan data tidak rusak. Maka baik terjadi kegagalan atau tidak koneksi ke database harus ditutup.
- Hal tersebut juga bisa terjadi saat pembacaan *file*. Untuk mencegah kerusakan *file*, baik akan terjadi *error* atau tidak, *file* harus ditutup. Di blok **finally** ini penanganan yang terjadi ketika *exception* muncul atau tidak disimpan.

Menggunakan “finally” pada Try-Except

Kode Program *exception9.py*

```
try:
    angka1 = float(input('masukkan angka ke-1 : '))
    angka2 = float(input('masukkan angka ke-2 : '))
    try:
        print('hasil perhitungan : ', angka1 / angka2)
    except ZeroDivisionError as e:
        print("proses perhitungan gagal karena : ", e)

except ValueError as e:
    print("proses input gagal karena : ", e)
finally:
    print("coba perhatikan lagi nilai yang anda masukkan ")

print("proses cetak ini masih dapat dijalankan ")
```

Menggunakan “finally” pada Try-Except

Kode Program *exception9.py* jika dieksekusi akan muncul tampilan seperti berikut :

```
masukkan angka ke-1 : 10
masukkan angka ke-2 : 20
hasil perhitungan :  0.5
coba perhatikan lagi nilai yang anda masukkan
proses cetak ini masih dapat dijalankan
```

```
masukkan angka ke-1 : 10
masukkan angka ke-2 : 0
proses perhitungan gagal karena :  float division by zero
coba perhatikan lagi nilai yang anda masukkan
proses cetak ini masih dapat dijalankan
```

With *statement*

- Pernyataan with dalam Python digunakan untuk manajemen sumber daya dan penanganan pengecualian (*exception handling*).
- Kemungkinan besar kita akan menemukannya saat bekerja dengan aliran file. Misalnya, pernyataan memastikan bahwa proses aliran file tidak memblokir proses lain jika *exception* muncul, tetapi berakhir dengan benar.

Menggunakan “with” statement

Blok kode di bawah ini menunjukkan pendekatan *try-finaly* untuk manajemen sumber daya aliran file.

```
file = open('file-path', 'w')
try:
    file.write('Lorem ipsum')
finally:
    file.close()
```

Biasanya, Anda ingin menggunakan metode ini untuk menulis ke file, tetapi pernyataan **with** menawarkan pendekatan yang lebih bersih:

```
with open('file-path', 'w') as file:
    file.write('Lorem ipsum')
```

Menggunakan “with” statement

Pernyataan with menyederhanakan proses penulisan kode menjadi hanya dua baris.

Ini juga digunakan dalam proses CRUD database. Contohnya sbb:

```
def get_all_songs():
    with sqlite3.connect('db/songs.db') as connection:
        cursor = connection.cursor()
        cursor.execute("SELECT * FROM songs ORDER BY id desc")
        all_songs = cursor.fetchall()
        return all_songs
```

Di sini, with digunakan untuk mengkueri database SQLite dan mengembalikan kontennya.

Menggunakan “with” statement

```
# a simple file writer object

class MessageWriter(object):
	def __init__(self, file_name):
		self.file_name = file_name

	def __enter__(self):
		self.file = open(self.file_name, 'w')
		return self.file

	def __exit__(self):
		self.file.close()

# using with statement with MessageWriter

with MessageWriter('my_file.txt') as xfile:
	xfile.write('hello world')
```

Menggunakan “with” statement

```
from contextlib import contextmanager

class MessageWriter(object):
    def __init__(self, filename):
        self.file_name = filename

    @contextmanager
    def open_file(self):
        try:
            file = open(self.file_name, 'w')
            yield file
        finally:
            file.close()

# usage with statement
message_writer = MessageWriter('hello.txt')
with message_writer.open_file() as my_file:
    my_file.write('hello world')
```

Tugas

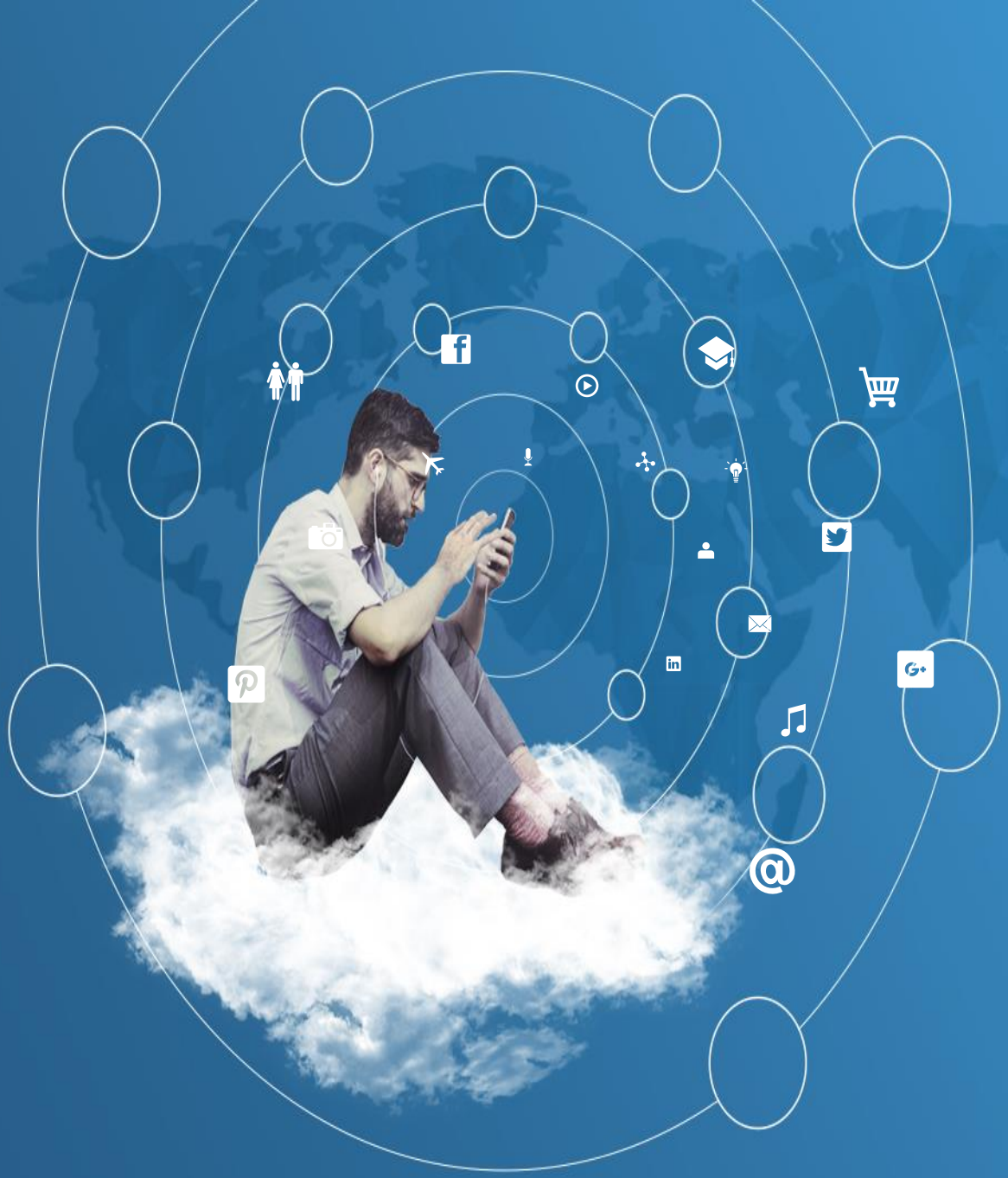
Lakukan percobaan pada VS Code untuk Program exception3.py, exception7.py, dan exception8.py.

Jawaban tugas ini adalah :

1. Screenshoot capture pada VScode program python yang anda ketik.
2. Screenshoot capture pada VScode hasil running program tsb pada window terminal VScode.

Hasil screenshoot bisa di rubah menjadi format pdf atau apapun sehingga sedemikian resolusi atau ukuran file total tidak melebihi 2MB seperti yang dipersyaratkan ketika mengupload jawaban anda.

Thank You



Programming for Science

Membuat File

Pengenalan File

- Biasanya jika kita tidak menggunakan *file*, hasil pemrosesan data hanya akan disimpan di *main memory*. Setelah program dihentikan atau tiba – tiba komputer Anda mati, semua data akan hilang.
- Lalu bagaimana jika ingin menggunakan kembali data yang sudah diproses sebelumnya ?.
- Untuk menyimpan data agar bisa diproses di kesempatan selanjutnya, misal komputer dimatikan kemudian dinyalakan lagi hari esoknya. Kita butuh sebuah penyimpanan yang bersifat *resident* dan disimpan di *secondary storage* seperti *harddisk*.
- Python sendiri menyediakan beberapa media penyimpanan yang bisa digunakan oleh programmer Python, ada ***file***, ***shelve***, ***marshal***, ***pickle***, dan ***sqlite3***.

Pengenalan File

- Pada bab ini kita akan bahas mengenai *file* berupat txt. *File* di Python bisa berupa txt, csv, atau jenis lainnya.
- Txt merupakan contoh file yang sering digunakan. *File* jenis ini berisi *plain text*. *File* jenis ini menyimpan karakter *ascii standard* yang diterima dari *user*.
- Pada pembuatan *file* terdapat beberapa mode dalam manipulasi *file*. Berikut daftar mode manipulasi *file* tersebut :

Pengenalan File

No	Mode	Keterangan
1	**r**	Membuka *file* dan hanya untuk pembacaan saja. *Pointer file* akan ditempatkan di awal *file*. Jika pada saat pembukaan *file* tidak disertakan mode manipulasi *file*, maka mode ini secara *default* dipakai untuk manipulasi *file*.
2	**w**	Membuka *file* dan hanya untuk penulisan saja. Jika *file* yang dibuka sudah ada dan menggunakan mode 'w', maka *file* tersebut akan ditimpa. Jika *file* tidak ada maka akan dibuatkan *file* baru.
3	**a**	Membuka *file* untuk penambahan isi *file*. *Pointer file* disimpan di bagian akhir *file* jika *file* tersebut ada. Jika *file* tidak ada maka akan dibuatkan *file* baru.
4	**b**	Mode ini ditambahkan masing – masing pada mode r, w, a menjadi rb, wb, ab. Dengan menambahkan mode b pada setiap mode manipulasi standar. Maka pembacaan *file* akan dilakukan dalam format biner
5	**+**	Mode ini ditambahkan kedalam mode r, w, a. • r+ : baca dan tulis • w+ : tulis dan baca • a+ : tambah dan baca

Membuat File Baru

- Agar lebih paham kita akan mencoba membuat sebuah *file* txt.
- Pertama kita biasakan cegah *error* dengan menggunakan *try-except* dan tangkap *exception* dengan *IOError*, agar jika terjadi *error* kelak, kita bisa menanganinya dengan lebih mudah.
- Kemudian di dalam blok *try-except* tersebut buat sebuah objek *file* dengan menggunakan *built-in function* **open()**.
- Pada *function* tersebut terdapat dua parameter yang biasa digunakan yaitu, nama *file* serta mode manipulasi *file*-nya. Karena kita ingin membuat *file* baru, maka mode “w” digunakan pada kasus pertama berikut :

Membuat File Baru

Kode Program *file1.py*

```
try:
    sebuah_file = open("absen.txt", 'w')
    print("nama file yang tadi dibuat : ", sebuah_file.name)
    print("mode pembacaan file : ", sebuah_file.mode)
    print("apakah filenya udah ditutup ? : ", sebuah_file.closed)
    sebuah_file.close()
    print("apakah filenya udah ditutup ? : ", sebuah_file.closed)
except IOError as e:
    print("proses gagal karena : ", e)
```

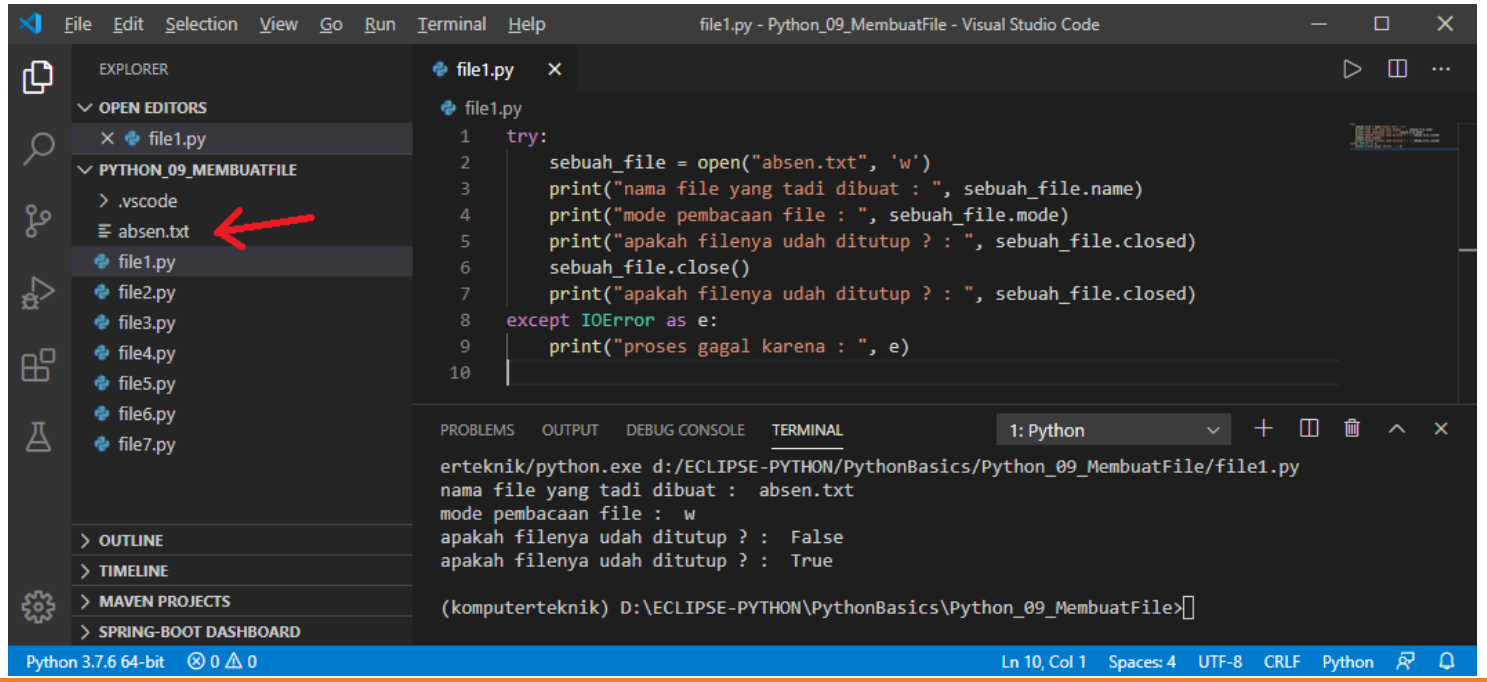
Membuat File Baru

- Ketika kita sudah membuka sebuah *file* dan terbentuk objek *file*. Kita dapat mengakses method dan atribut pada objek *file* tersebut.
- Atribut yang sering diakses untuk pemrosesan *file* antara lain : **name, mode, closed**. Atribut **name** adalah nama *file* tersebut, **mode** adalah mode manipulasi *file* tersebut, dan **closed** menyatakan apakah *file* tersebut sudah ditutup atau belum.
- Sedangkan *method* yang diakses diatas adalah **close()**, yang digunakan untuk menutup *file* setelah penggunaan *file* selesai. Dengan menutup *file*, penggunaan memori utama akan dihemat. Jika tidak pernah menutup *file* dalam jumlah yang banyak bisa menyebabkan *memory leak*. Jadi berhati – hatilah.

Membuat File Baru

Kode Program *file1.py* jika dieksekusi akan muncul tampilan seperti berikut :

```
nama file yang tadi dibuat :  absen.txt
mode pembacaan file :  w
apakah filenya udah ditutup ? :  False
apakah filenya udah ditutup ? :  True
```



Mengisi File

- Pada contoh pertama, *file* yang kita buat masih kosong, belum berisi. Sesuai namanya kita sedang membuat *file* bernama **'absen.txt'**, yang didalamnya akan terdapat daftar hadir perkuliahan.
- Dengan menggunakan *method* **write()**, kita bisa menambahkan isi pada *file* 'absen.txt', dan yang akan kita isikan adalah teks.
- Method ini memerlukan parameter sebuah string yang akan ditulis di lokasi tertentu pada *file* berdasarkan posisi *pointer file* berada.

Mengisi File

Kode Program *file2.py*

```
try:
    sebuah_file = open("absen.txt", 'w')
    print("nama file yang tadi dibuat : ", sebuah_file.name)
    print("mode pembacaan file : ", sebuah_file.mode)
    print("apakah filenya udah ditutup ? : ", sebuah_file.closed)
    sebuah_file.write('1. Jajang Surahman, Teknik Informatika,
ISTN\n')
    sebuah_file.write('2. Angel Corine, Sipil, ISTN\n')
    sebuah_file.write('3. Samsul Basri, Elektro, ISTN\n')
    sebuah_file.close()
    print("apakah filenya udah ditutup ? : ", sebuah_file.closed)
except IOError as e:
    print("proses gagal karena : ", e)
```

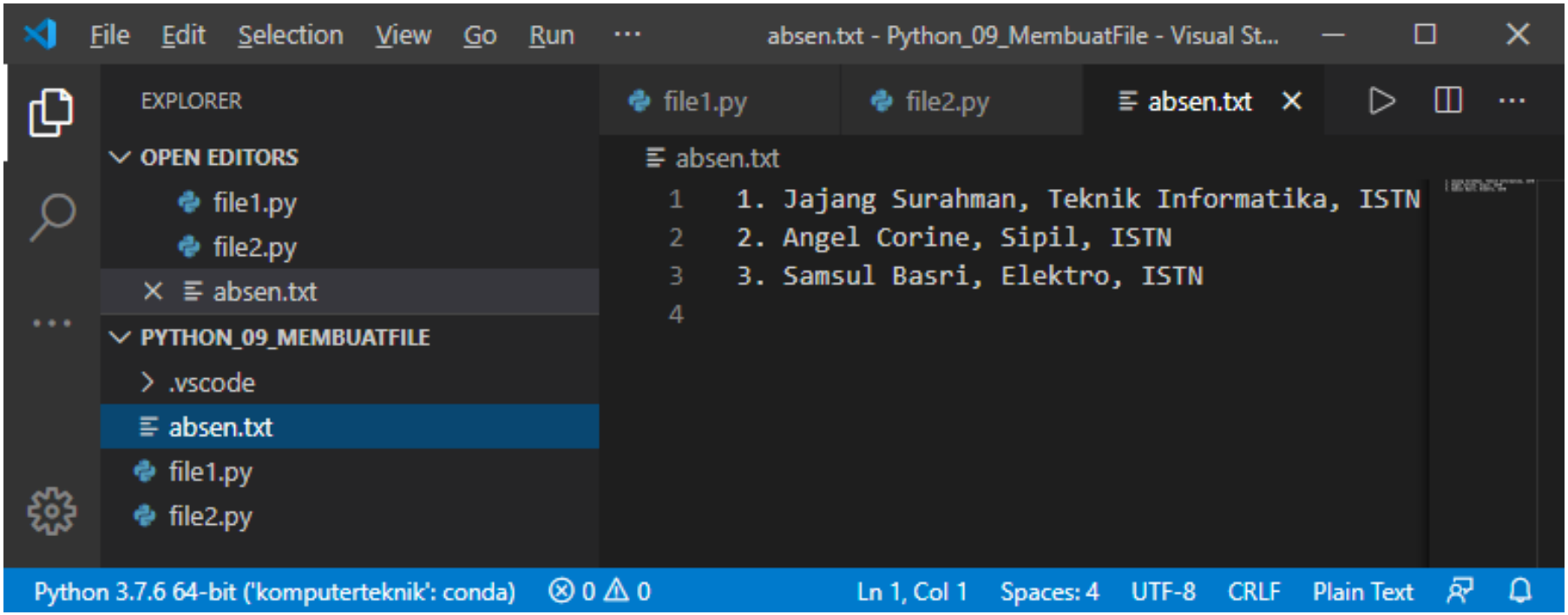
Mengisi File

Setelah kita menambahkan isi pada *file* teks yang kita buat, kita dapat membuka *file* yang telah dibuat dengan teks editor dan dapat melihat isi dari *file* tersebut. *Output* Program *file2.py* sbb:

```
nama file yang tadi dibuat :  absen.txt
mode pembacaan file :  w
apakah filenya udah ditutup ? :  False
apakah filenya udah ditutup ? :  True
```

Mengisi File

Isi file absen.txt yang dibuka dengan VSCode.



Membaca Isi File

- Setelah mengisi *file* dengan *method* **write()**. Sekarang kita akan menggunakan *method* **read()** untuk membaca *file*.
- Pastikan, *file* yang akan dibaca harus dalam mode **'r'**, jika tidak dalam mode tersebut, misal dalam mode **'w'**, maka akan muncul error : ”OError: File not open for reading”.
- Kemudian untuk mengetahui posisi *pointer file* berada, kita gunakan method **tell()**.

Membaca Isi File

Kode Program *file3.py*

```
try:
    sebuah_file = open("absen.txt", 'r')
    print("nama file yang tadi dibuat : ", sebuah_file.name)
    print("mode pembacaan file : ", sebuah_file.mode)
    print("apakah filenya udah ditutup ? : ", sebuah_file.closed)
    print("isi file : \n", sebuah_file.read())
    print("posisi pointer pada file : ", sebuah_file.tell())
    sebuah_file.close()
    print("apakah filenya udah ditutup ? : ", sebuah_file.closed)
except IOError as e:
    print("proses gagal karena : ", e)
```

Membaca Isi File

Output Program *file3.py* sbb:

```
nama file yang tadi dibuat :  absen.txt
mode pembacaan file :  r
apakah filenya udah ditutup ? :  False
isi file :
 1. Jajang Surahman, Teknik Informatika, ISTN
2. Angel Corine, Sipil, ISTN
3. Samsul Basri, Elektro, ISTN

posisi pointer pada file :  108
apakah filenya udah ditutup ? :  True
```

Membaca Isi File

- Dengan menggunakan *method* **read()**, kita dapat melihat isi dari *file* tersebut.
- Tapi *method* ini membaca sekaligus isi *file* yang dibaca, tidak perbaris. Jika pembacaan dilakukan sekaligus, ruang memori yang dibutuhkan jauh lebih besar daripada *file* yang dibaca perbaris.
- Kemudian dengan adanya *method* **tell()**, kita bisa mengetahui posisi *pointer file* berada dimana, agar mempermudah saat manipulasi *file*.

Membaca Isi File dengan cara Baris per Baris

- Jika pada contoh sebelumnya pembacaan *file* dilakukan sekaligus, pada contoh kali ini pembacaan *file* akan dilakukan baris perbaris.
- Pembacaan *file* teks dengan membaca perbaris ini bisa dilakukan dengan menggunakan pengulangan **for**.
- *File* ini diperlakukan layaknya **list** yang digunakan di pengulangan **for**. Disini *file* dianggap sebagai **list** yang berisi elemen *string*.

Membaca Isi File dengan cara Baris per Baris

Kode Program *file4.py*

```
try:
    sebuah_file = open("absen.txt", 'r')
    print("nama file yang tadi dibuat : ", sebuah_file.name)
    print("mode pembacaan file : ", sebuah_file.mode)
    print("apakah filenya udah ditutup ? : ", sebuah_file.closed)

    print("isi file : \n")

    for line in sebuah_file:
        print(line)

    print("posisi pointer pada file : ", sebuah_file.tell())
    sebuah_file.close()
    print("apakah filenya udah ditutup ? : ", sebuah_file.closed)

except IOError as e:
    print("proses gagal karena : ", e)
```

Membaca Isi File dengan cara Baris per Baris

Hasil yang diperlihatkan hampir sama dengan contoh sebelumnya hanya saja teknik pembacaannya sedikit berbeda. Jika file berukuran besar, akan lebih bijak jika kita membacanya perbaris agar ruang memori yang digunakan tidak banyak terpakai.

Output Program *file4.py* adalah seperti berikut :

```
nama file yang tadi dibuat :  absen.txt
mode pembacaan file :  r
apakah filenya udah ditutup ? :  False
isi file :

1. Jajang Surahman, Teknik Informatika, ISTN

2. Angel Corine, Sipil, ISTN

3. Samsul Basri, Elektro, ISTN

posisi pointer pada file :  108
apakah filenya udah ditutup ? :  True
```

Mengatur Posisi Pointer File

- Suatu saat kita ingin mengisi *file* di lokasi tertentu di sebuah *file*. Biasanya kita menambahkan konten *file* di bagian akhir *file*, atau membaca *file* selalu dibagian awal *file*. Ada saatnya kita ingin membaca di posisi ke 15 dari awal *file*, atau posisi -15 karakter dari *pointer file*. Di objek *file* terdapat *method* yang dinamakan **seek()**.
- *Method* tersebut memiliki dua buah paramater yaitu jarak yang diinginkan dan patokan jarak tersebut. Jika parameter kedua diisi oleh angka 0, berarti patokan berada di awal *file*. Jika parameter kedua diisi oleh angka 1, berarti patokan berada di tempat *pointer file* berada.
- Jika parameter kedua diisi oleh angka 2, maka patokan berada di bagian akhir *file*. Jika parameter pertama diisi angka positif maka penentuan jarak akan dihitung ke sebelah kanan, jika diisi angka negatif maka penentuan jarak akan dihitung ke sebelah kiri.

Mengatur Posisi Pointer File

Kode Program *file5.py*

```
try:
    sebuah_file = open("absen.txt", 'rb')

    print("nama file yang tadi dibuat : ", sebuah_file.name)
    print("mode pembacaan file : ", sebuah_file.mode)
    print("apakah filenya udah ditutup ? : ", sebuah_file.closed)

    print("isi file : \n")
    for line in sebuah_file:
        print(line)

    print("posisi pointer pada file : ", sebuah_file.tell())
    print("kembali lagi ke awal : ", sebuah_file.seek(0, 0))

    print("isi file : \n")
    for line in sebuah_file:
        print(line)

    print("posisi pointer pada file : ", sebuah_file.tell())
    sebuah_file.close()
    print("apakah filenya udah ditutup ? : ", sebuah_file.closed)

except IOError as e:
    print("proses gagal karena : ", e)
```

Mengatur Posisi Pointer File

Pada contoh diatas, pointer file dipindahkan kembali ke posisi awal. Dengan memberikan jarak 0, dan menentukan patokan di awal file, maka posisi pointer file pindah ke bagian awal file. Dengan demikian file bisa dibaca ulang untuk kedua kalinya

Output program *file5.py* adalah seperti berikut :

```
nama file yang tadi dibuat :  absen.txt
mode pembacaan file :  rb
apakah filenya udah ditutup ? :  False
isi file :

b'1. Jajang Surahman, Teknik Informatika, ISTN\r\n'
b'2. Angel Corine, Sipil, ISTN\r\n'
b'3. Samsul Basri, Elektro, ISTN\r\n'
posisi pointer pada file :  108
kembali lagi ke awal :  0
isi file :

b'1. Jajang Surahman, Teknik Informatika, ISTN\r\n'
b'2. Angel Corine, Sipil, ISTN\r\n'
b'3. Samsul Basri, Elektro, ISTN\r\n'
posisi pointer pada file :  108
apakah filenya udah ditutup ? :  True
```

Mengganti Nama File

- Dalam manipulasi *file*, terdapat operasi seperti pengubahan nama *file*, memindahkan *file*, ataupun menghapus *file*.
- Python sendiri menyediakan module os yang didalamnya terdapat fitur- fitur tersebut.
- Sebagai contoh pertama kita akan mengganti nama *file* dari “absen.txt” ke “daftarhadir.txt”. Pertama kita harus meng-import modul **os**. Kemudian kita gunakan *method* **rename()**.
- *Method* tersebut memiliki dua parameter yaitu nama *file* yang akan diubah namanya, dan nama baru yang diinginkan.

Mengganti Nama File

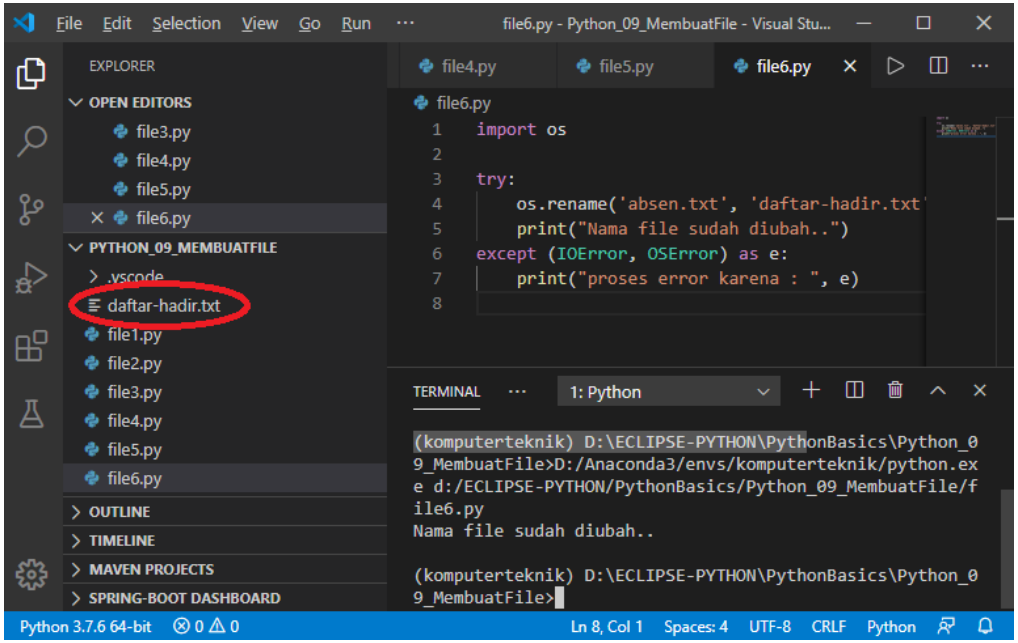
Kode Program *file6.py*

```
import os

try:
    os.rename('absen.txt', 'daftar-hadir.txt')
    print("Nama file sudah diubah..")
except (IOError, OSError) as e:
    print("proses error karena : ", e)
```

Mengganti Nama File

Output program *file6.py* adalah seperti berikut :

```
(komputerteknik) D:\ECLIPSE-
PYTHON\PythonBasics\Python_09_MembuatFile>D:/Anaconda3/envs/kompute
rteknik/python.exe d:/ECLIPSE-
PYTHON/PythonBasics/Python_09_MembuatFile/file6.py
Nama file sudah diubah..

(komputerteknik) D:\ECLIPSE-
PYTHON\PythonBasics\Python_09_MembuatFile>
```



Menghapus File

- Contoh lainnya adalah menghapus *file*. Kita bisa gunakan *method* **remove()** untuk menghapus *file* yang diinginkan.
- Parameter yang dibutuhkan adalah nama *file* yang akan dihapus.

Menghapus File

Kode Program *file7.py*

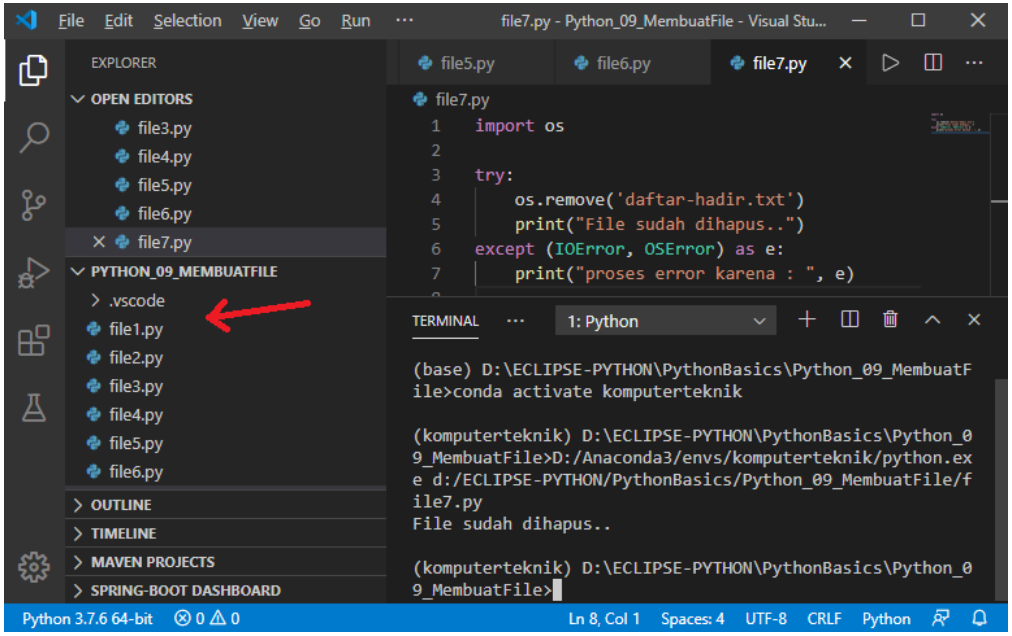
```
import os

try:
    os.remove('daftar-hadir.txt')
    print("File sudah dihapus..")
except (IOError, OSError) as e:
    print("proses error karena : ", e)
```

Kemudian jika file sudah dihapus, kita tidak dapat membuka file tersebut. Karena file tersebut sudah hilang dari penyimpanan sekunder

Menghapus File

Output program *file7.py* adalah seperti berikut :

```
(komputerteknik) D:\ECLIPSE-
PYTHON\PythonBasics\Python_09_MembuatFile>D:/Anaconda3/envs/kompute
rteknik/python.exe d:/ECLIPSE-
PYTHON/PythonBasics/Python_09_MembuatFile/file7.py
File sudah dihapus..

(komputerteknik) D:\ECLIPSE-
PYTHON\PythonBasics\Python_09_MembuatFile>
```



Pengelolaan File

Untuk contoh – contoh lainnya Anda bisa membuka dokumentasi resmi Python, atau coba kunjungi beberapa website seperti berikut.

http://www.tutorialspoint.com/python

http://www.zetcode.com/python

http://learnpythonthehardway.org/

http://pymotw.com

Thank You

Programming for Science

Pengenalan Class

Membuat Class dan Object

- Pemrograman berorientasi objek atau dalam bahasa inggris dikenal dengan *Object Oriented Programming* (OOP), merupakan sebuah paradigma pemrograman dimana kita memodelkan perangkat lunak kita dari berbagai kumpulan objek yang saling berinteraksi.
- Objek tersebut memiliki karakteristik dan aksi. Di dalam OOP, karakteristik tersebut berupa variabel yang dinamakan atribut. Kemudian aksi yang dimiliki objek tersebut berupa *method* yang menghasilkan *output* atau cuma melakukan aksi saja tanpa *output*. Ada istilah lain juga yang menyebut aksi sebuah objek sebagai perilaku atau *behaviour*.
- Objek itu sendiri mempunyai *template* yang diistilahkan dengan kelas atau *class*. Sebuah kelas merupakan *template* bagi objek – objek yang akan dibuat. Proses pembuatan objek baru dinamakan dengan instansiasi.

Membuat Class dan Object

- Ada beberapa hal yang harus diingat dalam membuat sebuah kelas. Pertama *keyword* **class**, *keyword* ini digunakan untuk mendefinisikan sebuah kelas. Disusul dengan nama kelas yang diinginkan dan tanda titik dua.
- Blok kode kelas tersebut ditulis setelah tanda titik dua tersebut, dan seperti biasa diperlukan indentasi agar blok kode yang kita tulis masuk kedalam blok kode kelas.
- Kedua adalah konstruktor, di Python konstruktor ditulis dengan sebuah *function* bernama **__init__()** .
- *Method* dan *function* sebenarnya sama, hanya saja beda istilah pada paradigma OOP.
- *Method* __init__() ini merupakan *method* yang akan selalu dieksekusi saat instansiasi objek.

Membuat Class dan Object

- Biasanya __init__() digunakan untuk mengisi variabel dengan nilai awal pada atribut – atribut yang dimiliki objek.
- Ketiga adalah *keyword* **self**, *keyword* tersebut digunakan pada *method* yang akan dinyatakan sebagai *method* dari kelas yang kita rancang.
- *Keyword* ini disertakan di parameter pertama. Jika *method* tersebut tidak disertakan **self** pada *method* yang dimiliki kelas tersebut, akan muncul error : "TypeError: nama_function() takes exactly n arguments (1 given)" yang artinya *method* tersebut tidak bisa dipanggil oleh objek yang telah terinstansiasi.

Membuat Class dan Object

Sebagai contoh disini kita akan membuat sebuah kelas bernama PersegiPanjang yang memiliki dua atribut yaitu panjang dan lebar. Kelas ini memiliki lima method yang terdiri dari :

- __init__(), konstruktor kelas persegi panjang.
- hitung_luas(), method untuk menghitung luas persegi panjang.
- hitung_keliling(), method untuk menghitung keliling persegi panjang.
- gambar_persegi_panjang(), menggambar persegi panjang yang direpresentasikan dengan kumpulan bintang.
- gambar_persegi_panjang_tanpa_isi(), menggambar persegi panjang tetapi hanya batas luarnya saja, isinya tak digambar.

Membuat Class dan Object

Kode Program *class1.py*

```
class PersegiPanjang:
    def __init__(self, panjang, lebar):
        self.panjang = panjang
        self.lebar = lebar

    def hitung_luas(self):
        return self.panjang * self.lebar

    def hitung_keliling(self):
        return (2*self.panjang) + (2*self.lebar)

    def gambar_persegi_panjang(self):
        for i in range(0, self.lebar):
            for j in range(0, self.panjang):
                print('*', end='')
            print("")

    def gambar_persegi_panjang_tanpa_isi(self):
        for i in range(0, self.lebar):
            if i > 0 and i < self.lebar-1:
                for j in range(0, self.panjang):
                    if j > 0 and j < self.panjang-1:
                        print(' ', end='')
                    else:
                        print('*', end='')
            else:
                for j in range(0, self.panjang):
                    print('*', end='')
            print("")
```

Membuat Class dan Object

Kode Program *class1.py* - lanjutan

```
PersegiPanjangA = PersegiPanjang(20, 10)
PersegiPanjangB = PersegiPanjang(10, 5)

print("Panjang persegi panjang A :", PersegiPanjangA.panjang)
print("Lebar persegi panjang A :", PersegiPanjangA.lebar)
print("Luas persegi panjang A : ", PersegiPanjangA.hitung_luas())
print("Keliling persegi panjang A : ", PersegiPanjangA.hitung_keliling())
print("Menggambar Persegi Panjang A : ")
PersegiPanjangA.gambar_persegi_panjang()

print("\nMenggambar Persegi Panjang A hanya tepinya saja : ")
PersegiPanjangA.gambar_persegi_panjang_tanpa_isi()

print("\n")

print("Panjang persegi panjang B :", PersegiPanjangB.panjang)
print("Lebar persegi panjang B :", PersegiPanjangB.lebar)
print("Luas persegi panjang B : ", PersegiPanjangB.hitung_luas())
print("Keliling persegi panjang B : ", PersegiPanjangB.hitung_keliling())
PersegiPanjangB.gambar_persegi_panjang()
print("\nMenggambar Persegi Panjang B hanya tepinya saja : ")
PersegiPanjangB.gambar_persegi_panjang_tanpa_isi()
```

Membuat Class dan Object

- Proses instansiasi dilakukan dengan menentukan nama objek yang diinginkan kemudian panggil nama kelas yang akan diinginkan dan masukan parameter awal yang dibutuhkan.
- Kemudian saat objek berhasil dibuat, kita bisa mengakses atribut dan method dari objek tersebut.
- Seperti yang kita lakukan pada manipulasi file, kita bisa mengakses *method* close(), write(), dan read() serta mengakses atribut **close**, **name**, dan **mode**. Dengan parameter berbeda namun karakterisitik sama, persegi panjang yang dihasilkan jadi beragam.

Membuat Class dan Object

- Tujuan dari OOP ini sendiri, menghemat penulisan kode program yang kita buat.
- Tanpa OOP kita harus membuat atribut untuk setiap objek. Dan penulisan kode program pun menjadi menumpuk, Karena untuk *method* yang sama harus di tulis ulang.

Membuat Class dan Object

Output program *class1.py* adalah seperti berikut :

```
Panjang persegi panjang A : 20
Lebar persegi panjang A : 10
Luas persegi panjang A :  200
Keliling persegi panjang A :  60
Menggambar Persegi Panjang A :
********************
********************
********************
********************
********************
********************
********************
********************
********************
********************

Menggambar Persegi Panjang A hanya tepinya saja :
********************
*                  *
*                  *
*                  *
*                  *
*                  *
*                  *
*                  *
*                  *
********************
```

Membuat Class dan Object

Output program *class1.py* adalah seperti berikut :

```
Panjang persegi panjang B : 10
Lebar persegi panjang B : 5
Luas persegi panjang B :  50
Keliling persegi panjang B :  30
**********
**********
**********
**********
**********

Menggambar Persegi Panjang B hanya tepinya saja :
**********
*         *
*         *
*         *
**********
```

Mengenal Built-in Function pada Class dan Object

Berikutnya kita akan mengenal beberapa *built-in class attribute* yang akan bisa digunakan saat kita membuat kelas apapun. *Built-in class attribute* akan selalu menyertai kelas yang kita rancang. Berikut beberapa atribut yang bisa Anda gunakan untuk mengakses informasi dari sebuah kelas :

- __doc__, digunakan untuk mengakses dokumentasi yang terdapat pada kelas
- __name__, digunakan untuk mengakses nama kelas
- __dict__, digunakan untuk mendapatkan namespace dari kelas tersebut. Kalau pada objek yang sudah diinstansiasi method ini akan mengeluarkan informasi tentang atribut yang sudah terisi nilai
- __module__, digunakan untuk mendapatkan informasi dimana lokasi modul yang mendefinisikan kelas tersebut
- __bases__, digunakan untuk melihat darimana kelas tersebut diwariskan. Pewarisan pada OOP adalah menggunakan karakteristik suatu kelas pada kelas yang ingin menggunakan karakteristik kelas yang mewariskannya.

Mengenal Built-in Function pada Class dan Object

Sebagai contoh ada beberapa *built-in class attribute* yang bisa diakses kelas dan objek hasil instansiasi dan ada yang hanya bisa diakeses kelas.

Mengenal Built-in Function pada Class dan Object

Kode Program *class2.py*

```
class PersegiPanjang:
    """
    Sebuah kelas yang memodelkan persegi panjang.
    Mempunyai dua atribut yaitu panjang dan lebar.
    Bisa menghitung luas dan keliling.
    Bisa juga menggambar persegi panjang sesuai atribut.
    """

    def __init__(self, panjang, lebar):
        self.panjang = panjang
        self.lebar = lebar

    def hitung_luas(self):
        return self.panjang * self.lebar

    def hitung_keliling(self):
        return (2*self.panjang) + (2*self.lebar)

    def gambar_persegi_panjang(self):
        for i in range(0, self.lebar):
            for j in range(0, self.panjang):
                print('*', end='')
            print("")
```

Mengenal Built-in Function pada Class dan Object

Kode Program *class2.py*

```
    def gambar_persegi_panjang_tanpa_isi(self):
        for i in range(0, self.lebar):
            if i > 0 and i < self.lebar-1:
                for j in range(0, self.panjang):
                    if j > 0 and j < self.panjang-1:
                        print(' ', end='')
                    else:
                        print('*', end='')
            else:
                for j in range(0, self.panjang):
                    print('*', end='')
            print("")


PersegiPanjangA = PersegiPanjang(20, 10)

print(PersegiPanjang.__doc__)
print(PersegiPanjang.__name__)
print(PersegiPanjang.__dict__)
print(PersegiPanjang.__module__)
print(PersegiPanjang.__bases__)

print(PersegiPanjangA.__doc__)
print(PersegiPanjangA.__dict__)
print(PersegiPanjangA.__module__)
```

Mengenal Built-in Function pada Class dan Object

- Pada contoh diatas, atribut __name__ dan __bases__ hanya bisa diakses oleh kelas. Sedangkan objek hasil instansiasi tidak bisa mengaksesnya.

Mengenal Built-in Function pada Class dan Object

Output program *class2.py* adalah seperti berikut :

```
    Sebuah kelas yang memodelkan persegi panjang.
    Mempunyai dua atribut yaitu panjang dan lebar.
    Bisa menghitung luas dan keliling.
    Bisa juga menggambar persegi panjang sesuai atribut.

PersegiPanjang
{'__module__': '__main__', '__doc__': '\n    Sebuah kelas yang memodelkan persegi panjang.\n
Mempunyai dua atribut yaitu panjang dan lebar.\n    Bisa menghitung luas dan keliling.\n
Bisa juga menggambar persegi panjang sesuai atribut.\n    ', '__init__': <function
PersegiPanjang.__init__ at 0x0000028E76214048>, 'hitung_luas': <function
PersegiPanjang.hitung_luas at 0x0000028E7620EEE8>, 'hitung_keliling': <function
PersegiPanjang.hitung_keliling at 0x0000028E7620EE58>, 'gambar_persegi_panjang': <function
PersegiPanjang.gambar_persegi_panjang at 0x0000028E7620EDC8>,
'gambar_persegi_panjang_tanpa_isi': <function PersegiPanjang.gambar_persegi_panjang_tanpa_isi
at 0x0000028E7620EF78>, '__dict__': <attribute '__dict__' of 'PersegiPanjang' objects>,
'__weakref__': <attribute '__weakref__' of 'PersegiPanjang' objects>}
__main__
(<class 'object'>,)

    Sebuah kelas yang memodelkan persegi panjang.
    Mempunyai dua atribut yaitu panjang dan lebar.
    Bisa menghitung luas dan keliling.
    Bisa juga menggambar persegi panjang sesuai atribut.

{'panjang': 20, 'lebar': 10}
__main__
```

Mengenal Built-in Function pada Class dan Object

- Pembahasan mengenai OOP (*Object Oriented Programming*) ini tidak bisa dibahas secara keseluruhan dalam perkuliahan Python dasar ini.
- Ada banyak hal yang harus diulas seperi *inheritance*, *polymorphism*, *abstract*, *overriding*, *overload*, dan lain – lain.
- Jika berkeinginan untuk mengetahui lebih dalam tentang Python OOP, silahkan eksplorasi melalui sumber-sumber lain.

Thank You

Programming for Science

Pengenalan Class Lanjut

Istilah Dalam OOP

Istilah	Penjelasan
Class	Prototipe yang ditentukan pengguna untuk objek yang mendefinisikan seperangkat atribut yang menjadi ciri objek kelas apa pun. Atribut adalah data anggota (variabel kelas dan variabel contoh) dan metode, diakses melalui notasi titik.
Class variable	Sebuah variabel yang dibagi oleh semua contoh kelas. Variabel kelas didefinisikan dalam kelas tapi di luar metode kelas manapun. Variabel kelas tidak digunakan sesering variabel contoh.
Data member	Variabel kelas atau variabel contoh yang menyimpan data yang terkait dengan kelas dan objeknya.
Function overloading	Penugasan lebih dari satu perilaku ke fungsi tertentu. Operasi yang dilakukan bervariasi menurut jenis objek atau argumen yang terlibat.
Instance variable	Variabel yang didefinisikan di dalam sebuah metode dan hanya dimiliki oleh instance kelas saat ini.
Inheritance	Pengalihan/pewarisan karakteristik kelas ke kelas lain yang berasal darinya.

Istilah Dalam OOP

Istilah	Penjelasan
Instance	Objek individu dari kelas tertentu. Istilah lain dari objek suatu kelas. Obyek obj yang dibuat dari prototipe kelas disebut sebagai instance dari kelas tersebut.
Instantiation	Penciptaan sebuah instance dari sebuah kelas.
Method	Jenis fungsi khusus yang didefinisikan dalam definisi kelas.
Object	Contoh unik dari struktur data yang didefinisikan oleh kelasnya. Objek terdiri dari kedua anggota data (variabel kelas dan variabel contoh) dan metode.
Operator overloading	Penugasan lebih dari satu fungsi ke operator tertentu.

Contoh Kelas Karyawan

Kode Program K*aryawan.py*

```
class Karyawan:
    '''Dasar kelas untuk semua karyawan'''
    jumlah_karyawan = 0

    def __init__(self, nama, gaji):
        self.nama = nama
        self.gaji = gaji
        Karyawan.jumlah_karyawan += 1

    def tampilkan_jumlah(self):
        print("Total karyawan:", Karyawan.jumlah_karyawan)

    def tampilkan_profil(self):
        print("Nama :", self.nama)
        print("Gaji :", self.gaji)
        print()
```

Contoh Kelas Karyawan

Kode Program *Karyawan.py* - lanjutan

```
# mencetak dokumentasi di kelas Karyawan
print(Karyawan.__doc__)

# Membuat objek pertama dari kelas Karyawan
karyawan1 = Karyawan("Sarah", 1000000)
# Membuat objek kedua dari kelas Karyawan
karyawan2 = Karyawan("Budi", 2000000)

karyawan1.tampilkan_profil()
karyawan2.tampilkan_profil()
print("Total karyawan :", Karyawan.jumlah_karyawan)
```

Contoh Kelas Karyawan

- Variabel jumlah_karyawan adalah variabel kelas yang dibagi ke semua instance/objek dari kelas ini. Variabel ini bisa diakses dari dalam atau luar kelas dengan menggunakan notasi titik, `Karyawan.jumlah_karyawan`.
- Metode `__init__()` adalah metode konstruktor, yaitu metode khusus yang digunakan Python untuk menginisialisasi pembuatan objek dari kelas tersebut.
- Fungsi – fungsi di dalam kelas (disebut metode) pendefinisiannya sama dengan fungsi pada umumnya. Hanya saja, harus ada argumen pertama bernama self. Pada saat pemanggilan fungsi, argumen ini otomatis ditambahkan oleh Python. Anda tidak perlu menambahkannya pada saat memanggil fungsi.

Instansiasi Objek

- Untuk mengakses dokumentasi yang ada di kelas dilakukan melalui perintah :

```
print(Karyawan.__doc__)
```

- Untuk membuat objek dari sebuah kelas, kita bisa memanggil nama kelas dengan argumen sesuai dengan fungsi __init__() pada saat kita mendefinisikannya.

```
# Membuat objek pertama dari kelas Karyawan
karyawan1 = Karyawan("Sarah", 1000000)
# Membuat objek kedua dari kelas Karyawan
karyawan2 = Karyawan("Budi", 2000000)
```

Mengakses Attribut Objek

- Kita bisa mengakses atribut objek dengan menggunakan operator titik. Variabel kelas bisa diakses dengan menggunakan nama kelasnya.

```
karyawan1.tampilkan_profil()
karyawan2.tampilkan_profil()
print("Total karyawan :", Karyawan.jumlah_karyawan)
```

Output Contoh Kelas Karyawan

Output Program *Karyawan.py* - lanjutan

```
Dasar kelas untuk semua karyawan
Nama : Sarah
Gaji : 1000000

Nama : Budi
Gaji : 2000000

Total karyawan : 2
```

Menambah, Menghapus, dan Mengubah Atribut Objek

- Kita bisa menambah, menghapus, dan mengubah atribut objek seperti berikut:

```
karyawan1.gaji = 1500000
karyawan1.nama = 'Ratna'
del karyawan1.gaji
```

- Cara yang lebih elegan untuk memodifikasi atribut adalah dengan menggunakan fungsi – fungsi berikut:
  - **getattr(obj, name[, default])** – Mengakses atribut objek
  - **hasattr(obj, name)** – Memeriksa apakah objek memiliki atribut tertentu atau tidak
  - **setattr(obj, name, value)** – Mengatur nilai atribut. Jika atribut tidak ada, maka atribut tersebut akan dibuatkan
  - **delattr(obj, name)** – Menghapus atribut dari objek

Menambah, Menghapus, dan Mengubah Atribut Objek

```
hasattr(karyawan1, 'gaji')   # True jika atribut
'gaji' ada

getattr(karyawan1, 'gaji')   # mengembalikan nilai
dari attribut 'gaji'

setattr(karyawan1, 'gaji', 1600000)  # mengatur
nilai atribut 'gaji'

delattr(karyawan1, 'gaji') # menghapus atribut
'gaji'
```

Atribut Kelas Built-in

- Setiap kelas di Python memiliki atribut built-in (bawaan) yang bisa diakses menggunakan operator titik. Attribut-attribut tersebut adalah sebagai berikut:
  - **__dict__** – dictionary yang berisi namespace dari kelas
  - **__doc__** – mengakses string dokumentasi (docstring) dari kelas
  - **__name__** – nama kelas
  - **__module__** – nama modul tempat kelas didefinisikan. Nilai attribut ini di mode interaktif adalah “__main__“.
  - **__bases__** – dasar dari kelas, bila kelas tidak merupakan turunan dari kelas lain, maka induknya dalah kelas object.

Atribut Kelas Built-in

Kode program *Karyawan2.py*

```
class Karyawan:
    '''Dasar kelas untuk semua karyawan'''
    jumlah_karyawan = 0

    def __init__(self, nama, gaji):
        self.nama = nama
        self.gaji = gaji
        Karyawan.jumlah_karyawan += 1

    def tampilkan_jumlah(self):
        print("Total karyawan:", Karyawan.jumlah_karyawan)

    def tampilkan_profil(self):
        print("Nama :", self.nama)
        print("Gaji :", self.gaji)
```

Atribut Kelas Built-in

Kode program *Karyawan2.py*

```
# Membuat objek pertama dari kelas Karyawan
karyawan1 = Karyawan("Sarah", 1000000)
# Membuat objek kedua dari kelas Karyawan
karyawan2 = Karyawan("Budi", 2000000)

print("Karyawan.__doc__:", Karyawan.__doc__)
print("Karyawan.__name__:", Karyawan.__name__)
print("Karyawan.__module__:", Karyawan.__module__)
print("Karyawan.__dict__:", Karyawan.__dict__)
print("Karyawan.__bases__:", Karyawan.__bases__)
```

Atribut Kelas Built-in

Output program *Karyawan2.py*

```
Karyawan.__doc__: Dasar kelas untuk semua karyawan
Karyawan.__name__: Karyawan
Karyawan.__module__: __main__
Karyawan.__dict__: {'__module__': '__main__', '__doc__':
'Dasar kelas untuk semua karyawan', 'jumlah_karyawan': 2,
'__init__': <function Karyawan.__init__ at
0x000002634B87A318>, 'tampilkan_jumlah': <function
Karyawan.tampilkan_jumlah at 0x000002634B8840D8>,
'tampilkan_profil': <function Karyawan.tampilkan_profil at
0x000002634B884048>, '__dict__': <attribute '__dict__' of
'Karyawan' objects>, '__weakref__': <attribute '__weakref__'
of 'Karyawan' objects>}
Karyawan.__bases__: (<class 'object'>,)
```

Penghancuran Objek (Pengumpulan Sampah/Garbage Collection)

- Python menghapus objek yang sudah tidak terpakai secara otomatis untuk menghemat memori. Proses ini disebut dengan pengumpulan sampah (garbage collection).
- Kolektor sampah Python terus berjalan pada saat program dieksekusi dan dipicu pada saat tidak ada lagi referensi/variabel yang merujuk ke objek.
- Jumlah referensi terhadap objek bertambah pada saat ada variabel yang merujuk ke objek tersebut. Sebaliknya referensi terhadap objek berkurang ketika variabel terhapus dengan menggunakan del, atau saat terjadi penugasan ulang, atau saat referensi keluar dari scope-nya.

Penghancuran Objek (Pengumpulan Sampah/Garbage Collection)

- Pada saat referensi terhadap objek sudah nol, maka Python akan otomatis menghapus objek tersebut. Perhatikan contoh berikut:

```
a = 30     # Menciptakan objek <30>
b = a      # menambah jumlah referensi ke objek <30>
c = [b]    # menambah jumlah referensi ke objek <30>

del a      # mengurangi jumlah referensi ke objek <30>
b = 100    # mengurangi jumlah referensi ke objek <30>
c[0] = -1 # mengurangi jumlah referensi ke objek <30>
```

- Pada contoh di atas, objek 30 pada akhirnya akan dihapus karena sudah tidak ada variabel yang merujuk ke objek tersebut.
- Python melakukan penghapusan objek secara otomatis tanpa ada pemberitahuan. Kita bisa menggunakan sebuah metode khusus yaitu metode `__del__()` yang disebut destruktor, yang akan dipanggil apabila sebuah objek akan dihapuskan oleh python.

Penghancuran Objek (Pengumpulan Sampah/Garbage Collection)

- Pada saat referensi terhadap objek sudah nol, maka Python akan apabila sebuah objek akan dihapuskan oleh python.
- Berikut adalah contoh penggunaan destruktor `__del__()`. Kode program *destruktordel.py*

Penghancuran Objek (Pengumpulan Sampah/Garbage Collection)

Kode program *DestruktorDel.py*

```
class Point:
    def __init__(self, x=0, y=0):
        self.x = x
        self.y = y

    def __del__(self):
        class_name = self.__class__.__name__
        print(class_name, "dihancurkan")


pt1 = Point()
pt2 = pt1
pt3 = pt1
print(id(pt1), id(pt2), id(pt3))  # ,menampilkan id objek
del pt1
del pt2
del pt3
```

Penghancuran Objek (Pengumpulan Sampah/Garbage Collection)

Output program *destruktordel.py*

```
2837074292616 2837074292616 2837074292616
Point dihancurkan
```

Pewarisan (Inheritansi) Kelas

- Kita bisa menurunkan karakteristik sebuah kelas ke kelas baru, dibandingkan dengan membuat kelas baru dari awal. Turunannya disebut kelas anak (child class) dan yang mewariskannya disebut kelas induk (parent class).
- Kelas anak mewarisi atribut dari kelas induk, dan kita bisa menggunakan atribut tersebut seolah atribut itu didefinisikan juga di dalam kelas anak. Kelas anak juga bisa menimpa (override) data dan metode dari induknya dengan data dan metodenya sendiri.
- Satu kelas anak bisa mewarisi karakteristik dari satu atau beberapa kelas induk.

Pewarisan (Inheritansi) Kelas

Sintaks

- Pewarisan memiliki sintaks sebagai berikut:

```
class SubClassName (ParentClass1[, ParentClass2, ...]):
    """docstring"""
    class_body
```

Pewarisan (Inheritansi) Kelas

Kode program *Turunan.py*

```
class Induk:  # mendefinisikan kelas Induk
    parent_attr = 100

    def __init__(self):
        print("Memanggil konstruktor induk")

    def parent_method(self):
        print('Memanggil metode induk')

    def set_attr(self, attr):
        Induk.parent_attr = attr

    def get_attr(self):
        print("Attribut induk :", Induk.parent_attr)

class Anak(Induk):  # mendefinisikan kelas Anak
    def __init__(self):
        print("Memanggil konstruktor Anak")

    def child_method(self):
        print('Memanggil metode Anak')
```

Pewarisan (Inheritansi) Kelas

Kode program *Turunan.py*

```
c = Anak()  # instansiasi kelas Anak
c.child_method()  # Anak memanggil metodenya

c.parent_method()  # Anak memanggil metode Induk
c.set_attr(200)  # Anak kembali memanggil metode
Induk
c.get_attr()  # anak kembali memanggil metode Induk
```

Output program *turunan.py*

```
Memanggil konstruktor Anak
Memanggil metode Anak
Memanggil metode induk
Attribut induk : 200
```

Pewarisan (Inheritansi) Kelas

- Dengan cara yang sama, kita bisa membuat mewariskan beberapa induk ke satu anak seperti berikut:

```
class A:      # mendefinisikan kelas A
.....

class B:      # mendefinisikan kelas B
.....

class C(A, B): # mendefinisikan turunan dari kelas A dan B
.....
```

Pewarisan (Inheritansi) Kelas

- Kita bisa menggunakan fungsi `issubclass()` atau `isinstance()` untuk memeriksa relasi antara dua kelas atau objek.
  - Fungsi `issubclass(sub, sup)` mengembalikan True jika sub merupakan anak dari sup
  - Fungsi `isinstance(obj, Class)` mengembalikan True jika obj adalah instance dari kelas Class atau subkelas dari Class.

Metode Overriding

- Kita bisa mengabaikan fungsi dari kelas induk di dalam kelas anak. Alasan untuk melakukan overriding adalah karena kita memodifikasi atau mengubah metode yang sudah diturunkan dari kelas induk di dalam kelas anak. Perhatikan contoh berikut:

Kode program *Overriding.py*

```
class Induk:
    def my_method(self):
        print("Ini my_method kelas induk")

class Anak(Induk):
    def my_method(self):
        print("Ini my_method kelas anak")

c = Anak()
c.my_method()
```

Perhatikan pada contoh ini bagaimana kita mengabaikan metode yang dari induk dan mendefinisikan sendiri metode dengan nama yang sama di kelas anak. Dan yang dijalankan adalah metode yang ada di kelas anak.

Output program *overriding.py*

```
Ini my_method kelas anak
```

Overloading Metode

- Tabel berikut menunjukkan beberapa fungsi umum yang sering di-override di dalam kelas:

No	Metode, Deskripsi, dan Contoh
1.	`__init__(self[, args…])` Fungsi: Konstruktor (argumen bersifat opsional) Contoh pemanggilan: obj = className(args)
2.	`__del__(self)` Fungsi: Destruktor, menghapus sebuah objek Contoh pemanggilan: del obj
3.	`__repr__(self)` Fungsi: Representasi string yang bisa dievaluasi Contoh pemanggilan: repr(obj)
4.	`__str__(self)` Fungsi: Representasi string yang bisa dicetak Contoh pemanggilan: str(obj)
5.	`__cmp__(self, x)` Fungsi: Membandingkan objek Contoh pemanggilan: cmp(obj, x)

Overloading Operator

- Misalkan kita membuat sebuah kelas Vector untuk menunjukkan vektor dua dimensi. Apa yang terjadi bila kita menggunakan tanda + untuk menjumlahkan keduanya?
- Kita bisa mendefinisikan metode __add__ dalam kelas kita untuk melakukan penjumlahan vektor dan kemudian operator + akan berfungsi sesuai kehendak kita. Perhatikan contoh berikut:

Overloading Operator

Kode program *OverloadingOperator.py*

```
class Vector:
    def __init__(self, a, b):
        self.a = a
        self.b = b

    def __str__(self):
        return 'Vector (%d, %d)' % (self.a, self.b)

    def __add__(self, other):
        return Vector(self.a + other.a, self.b + other.b)

v1 = Vector(5, 10)
v2 = Vector(4, -2)

print(v1 + v2)
```

Output program *OverloadingOperator.py*

```
Vector (9, 8)
```

Menyembunyikan Data (*Data Hiding*)

- Sebuah atribut objek bisa dibuat terlihat ataupun tersembunyi dari luar kelas. Caranya di Python adalah dengan memberi nama atribut dengan di awali tanda underscore dua kali. Dengan begitu, atribut tersebut tidak akan dapat tampak dari luar kelas.

Menyembunyikan Data (*Data Hiding*)

Kode program *DataHiding.py*

```
class Counter:
    __secret_count = 0

    def count(self):
        self.__secret_count += 1
        print(self.__secret_count)

counter = Counter()
counter.count()
counter.count()
print(counter.__secret_count)
```

Output program *DataHiding.py*

```
1
2
Traceback (most recent call last):
  File "d:/PROJECTS-PYTHON/PythonBasics/Python_10_ClassDanOOP/DataHiding.py", line
12, in <module>
    print(counter.__secret_count)
AttributeError: 'Counter' object has no attribute '__secret_count'
```

Menyembunyikan Data (*Data Hiding*)

- Python melindungi attribut tersebut dengan mengubah namanya. Kita bisa mengakses atribut seperti itu dengan format object._className__attrName seperti berikut:

Kode program *DataHiding2.py*

```
class Counter:
    __secret_count = 0

    def count(self):
        self.__secret_count += 1
        print(self.__secret_count)

counter = Counter()
counter.count()
counter.count()
print(counter._Counter__secret_count) # sembunyikan data
```

```
1
2
2
```

Tugas

Lakukan percobaan pada VS Code untuk Program Karyawan.py, Turunan.py, dan OverloadingOperator.py.

Jawaban tugas ini adalah :

1. Screenshoot capture pada VScode program python yang anda ketik.
2. Screenshoot capture pada VScode hasil running program tsb pada window terminal VScode.

Hasil screenshoot bisa di rubah menjadi format pdf atau apapun sehingga sedemikian resolusi atau ukuran file total tidak melebihi 2MB seperti yang dipersyaratkan ketika mengupload jawaban anda.

Thank You

Programming for Science

Pengenalan Module

Module dan Packages

- *Module* adalah istilah *file* yang berisi kode Python. Jadi dari awal sebenarnya kita sudah membuat *module* Python. Hanya saja pada konsep *module* ini, kode Python yang akan digunakan berulang akan dipisahkan dari *file* utama ke dalam *file* lain yang khusus menampung kode Python tersebut.
- Di dalam module kita bisa menyimpan *class*, *function*, variabel, dan struktur data yang bisa digunakan oleh program. Misal kita ingin membuat sebuah kode yang hanya berisi jenis – jenis segitiga seperti segitiga sama kaki, segitiga sembarang, segitiga sama sisi, dan segitiga siku – siku.
- Kenapa tidak dicampur saja dengan jenis bidang yang lain ? Hal ini dilakukan agar kita mudah dalam mengelola kode Python yang kita tulis. Contoh lainnya misal kita menulis kode yang berinteraksi dengan database dan kode untuk melakukan proses penulisan laporan secara terpisah.

Module dan Packages

- Dalam hal ini *module* mempunyai kode Python yang *reusable* agar kode yang ditulis pada program kita terduplikasi.
- Sedangkan *file* Python yang akan dijalankan dan memanggil *function*, *class*, atau variabel dari kumpulan module yang dibuat berisi *runnable code*.
- Kode yang dieksekusi oleh *interpreter* Python untuk menampilkan wujud dari program yang dibuat.

Module dan Packages

- Kemudian *module-module* yang sudah ditulis bisa dikelompokan kedalam sebuah *package*.
- *Package* ini sendiri berupa *folder* yang memiliki *file* __init__.py, agar *folder* tersebut dikenali sebagai *module*.
- Di dalam *package* ini *module-module* memiliki tujuan dan fungsional yang seragam.
- Misal pada contoh yang akan kita coba, terdapat sebuah *package* bidang, yang berisi *module* bidang segitiga dan persegi. Di dalamnya terdapat *file* __init__.py yang bertugas untuk *meload* semua *module* yang ada di dalam *package*, segitiga.py yang berisi *class* segitiga, dan persegi.py yang berisi *class* persegi.
- Di dalam *file* segitiga.py dan persegi.py masing – masing bisa diisi berbagai jenis bidang yang sesuai nama *module* tersebut. Hanya saja untuk contoh kali ini dibatasi kepada satu jenis bidang saja.

Membuat Module-Module di dalam Packages

- Setelah memahami konsep *module*, mari kita coba program yang agak banyak ini.
- Sebelumnya di direktori tempat kita akan menulis program, terlebih dahulu buatlah sebuah *folder* baru bernama bidang.
- *Folder* tersebut merupakan *package* yang akan menyimpan persegi.py, segitiga.py, dan __init__.py.

Membuat Module-Module di dalam Packages

Kode Program *persegi.py*

```
class Persegi:
    def __init__(self, s):
        self.sisi = s

    def SetSisi(self, s):
        self.sisi = s

    def GetSisi(self):
        return self.sisi

    def HitungKeliling(self):
        return 4 * self.sisi

    def HitungLuas(self):
        return self.sisi * self.sisi
```

Membuat Module-Module di dalam Packages

- Kode persegi.py diatas hanya berisi *class* Persegi yang mempunyai atribut sisi dan *method-method* nya.
- Di dalam *module* ini kita bisa saja menulis kelas PersegiPanjang. Namun hal tersebut tidak dilakukan agar memudahkan kita dimana bidang yang jenisnya persegi tidak tercampur dengan bidang yang jenisnya segitiga.
- Pastikan Anda menyimpan *file* persegi.py di dalam *folder* bidang.

Membuat Module-Module di dalam Packages

Kode Program *segitiga.py*

```
import math

class Segitiga:

    def __init__(self, a, t):
        self.alas = a
        self.tinggi = t

    def SetAlas(self, a):
        self.alas = a

    def GetAlas(self):
        return self.alas

    def SetTinggi(self, t):
        self.tinggi = t

    def GetTinggi(self):
        return self.tinggi

    def GetSisiMiring(self):
        return math.sqrt(self.alas**2 + self.tinggi**2)

    def HitungKeliling(self, s):
        return self.alas + self.tinggi + s

    def HitungLuas(self):
        return (self.alas * self.tinggi) / 2
```

Membuat Module-Module di dalam Packages

- Hampir sama dengan fungsi dari module persegi.py, hanya saja *module* segitiga.py akan diisi berbagai jenis segitiga.
- Selain itu pada kode diatas kita memanggil *module* **math** yag merupakan paket pustaka dari python, karena saat nanti *module* segitiga.py ini diload, kode yang menggunakan *method-method* pada **math** harus di load juga dari module **math**.
- Pastikan *module* ini tersimpan di *folder* bidang.

Membuat Module-Module di dalam Packages

Kode Program *__init__.py*

```
from bidang.segitiga import Segitiga
from bidang.persegi import Persegi

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

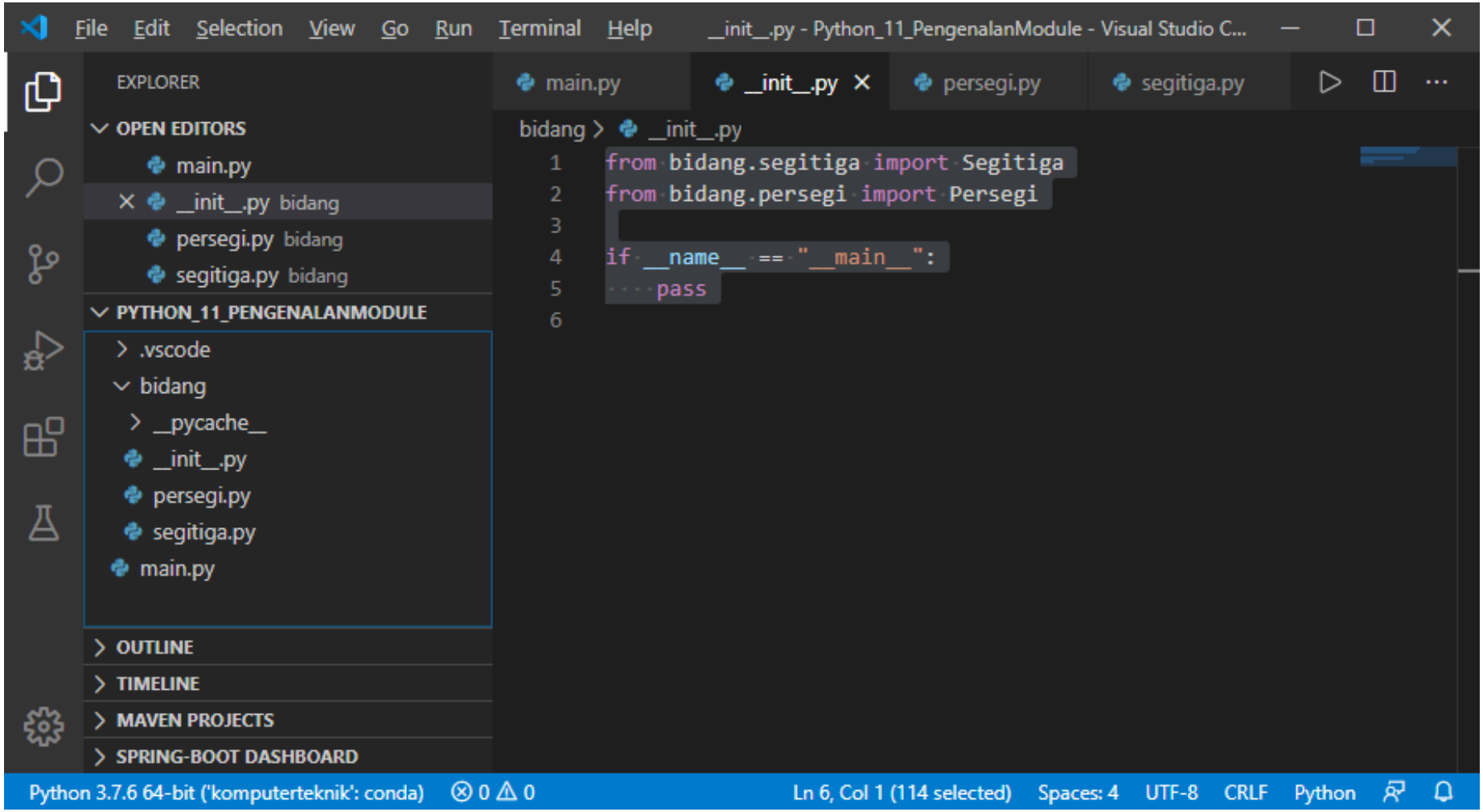
Membuat Module-Module di dalam Packages

- Kemudian *file* yang mesti ada di dalam sebuah *package* adalah __init__.py. *File* tersebut berfungsi untuk me-*load* isi *module* ke dalam memori agar isi *module* bisa digunakan di *file* yang berisi *runnable code*.
- Pada kode diatas, terdapat sintaks : **from** bidang.segitiga **import** Segitiga. *Keyword* **from** adalah *keyword* yang digunakan untuk menentukan *package* atau *module* mana yang akan kita rujuk, sedangkan **import** digunakan untuk mengambil *class*, *function* atau variabel yang didefinisikan di dalam *module*.
- Disana kita meng-*import* dua buah kelas yaitu Segitiga dan Persegi dari dua *module* berbeda yaitu segitiga.py dan persegi.py.
- Sedangkan kode dibawahnya digunakan jika file __init__.py ingin menjalankan perintah tertentu.
- Pastikan file ini disimpan di folder bidang.

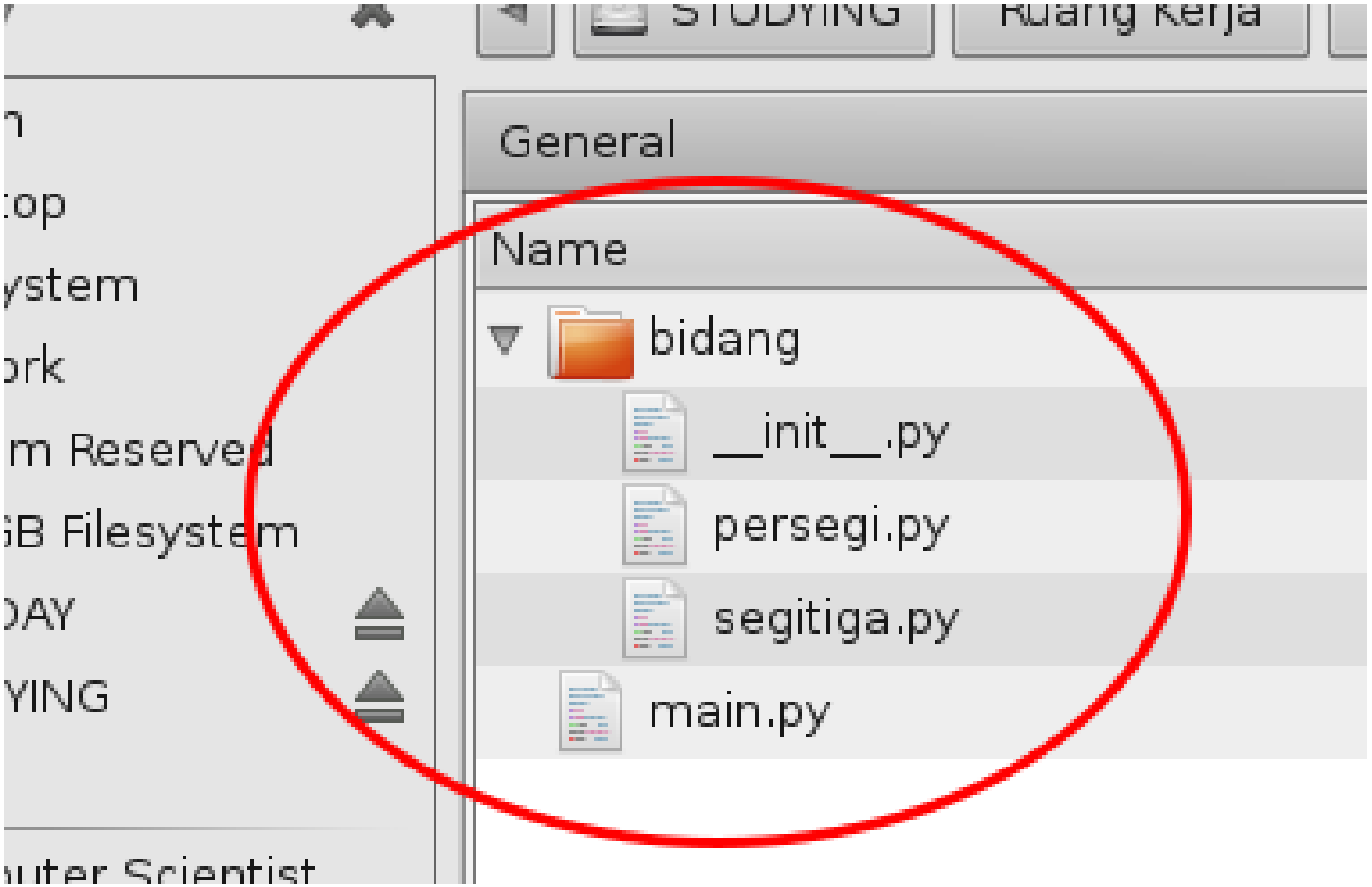
Menggunakan Module di File Utama

- Sampai akhirnya kita tiba untuk menulis kode utama.
- Kode utama ini merupakan kode yang berisi *runnable code*, dan menggunakan *class* yang sudah didefinisikan di *module-module* sebelumnya.
- Dengan demikian kode program tidak akan menumpuk di *file* utama.
- Jika Anda berhasil mengikuti petunjuk pada bab ini, *module*, *packages* dan *file* utama harus mempunyai sususan seperti berikut :

Menggunakan Module di File Utama



Menggunakan Module di File Utama



Menggunakan Module di File Utama

Kode Program *main.py*

```
from bidang import segitiga, persegi
from bidang.segitiga import Segitiga
from bidang.persegi import Persegi

sgtgA = Segitiga(3, 9)
prsgA = Persegi(5)

print("Luas Segitiga A : ", sgtgA.HitungLuas())
print("Sisi Miring Segitiga A : ", sgtgA.GetSisiMiring())
print("Keliling Segitiga A : ", sgtgA.HitungKeliling(sgtgA.G
etSisiMiring()))

print("\n")

print("Luas Persegi A : ", prsgA.HitungLuas())
print("Keliling Segitiga A : ", prsgA.HitungKeliling())
```

Menggunakan Module di File Utama

Pada kode diatas kita meng-*import* kelas dari *package* bidang. Kemudian melakukan instansiasi dan memberikan nilai sesuai yang kita inginkan.

Kemudian kita akses *method-method* yang terdapat pada kelas tersebut untuk mendapatkan informasi luas, dan keliling pada masing-masing bidang.

Jika berhasil maka kode yang akan dijalankan seperti berikut :

```
(komputerteknik) D:\ECLIPSE-
PYTHON\PythonBasics\Python_11_PengenalanModule>D:/Anaconda3/envs/komputertek
nik/python.exe d:/ECLIPSE-
PYTHON/PythonBasics/Python_11_PengenalanModule/main.py
Luas Segitiga A :  13.5
Sisi Miring Segitiga A :  9.486832980505138
Keliling Segitiga A :  21.486832980505138


Luas Persegi A :  25
Keliling Segitiga A :  20

(komputerteknik) D:\ECLIPSE-PYTHON\PythonBasics\Python_11_PengenalanModule>
```

Thank You

Programming for Science

Kasus-kasus penyelesaian dengan Python

Contoh Kasus^2 Komputasi dan Matematika dasar menggunakan Python

1. Program meminta input angka dari pengguna.
2. Program menentukan sebuah bilangan adalah bilangan prima atau tidak.
3. Program mencari faktor dari sebuah bilangan.
4. Program memecahkan persamaan kuadrat.

   Program memecahakn persamaan kuadrat dengan algoritma lain.
5. Program untuk mencari Faktor Persekutuan Besar (FPB) dari dua bilangan.
6. Program kalkulator sederhana (perkalian, pembagian, penambahan, pengurangan).

Contoh Kasus^2 Komputasi dan Matematika dasar menggunakan Python

7. Program mencetak semua permutasi bilangan.
8. Program mencetak semua permutasi dengan panjang r tertentu.
9. Program mencetak kombinasi dari list bilangan dengan panjang r tertentu.
10. Program mencetak kombinasi dari list bilangan dengan panjang r tertentu dan elemen bilangan yang sama.
11. Program menghitung zakat penghasilan.
12. Program menentukan nilai akhir semester (dalam 4 grade).

1. Menjumlahkan dua bilangan dengan input angka dari pengguna

Kode Program *Kasus1_InputAngkaPengguna.py*

```
# Meminta input dari pengguna
bil1 = input("Masukkan bilangan pertama: ")
bil2 = input('Masukkan bilangan kedua: ')

# Menjumlahkan bilangan
jumlah = float(bil1) + float(bil2)

# Menampilkan jumlah
print('Jumlah {0} + {1} adalah {2}'.format(bil1, bil2, jumlah))
```

Output *Kasus1_InputAngkaPengguna.py*

```
(komputerteknik) D:\Kasus1_InputAngkaPengguna.py
Masukkan bilangan pertama: 10
Masukkan bilangan kedua: 30
Jumlah 10 + 30 adalah 40.0
```

2. Menentukan sebuah bilangan termasuk bilangan prima atau tidak

Kode Program *Kasus2_TentukanPrima.py*

```
# Meminta input bilangan dari pengguna
num = int(input("Masukkan bilangan: "))

# bilangan prima harus lebih besar dari 1
if num > 1:
    for i in range(2, num):
        if (num % i) == 0:
            print(num, "bukan bilangan prima")
            print(i, "kali", num//i, "=", num)
            break
        else:
            print(num, "adalah bilangan prima")
            break

# bila bilangan kurang atau sama dengan satu
else:
    print(num, "bukan bilangan prima")
```

2. Menentukan sebuah bilangan termasuk bilangan prima atau tidak

Output *Kasus2_TentukanPrima.py*

```
(komputerteknik) D:\Kasus2_TentukanPrima.py
Masukkan bilangan: 17
17 adalah bilangan prima
```

```
(komputerteknik) D:\Kasus2_TentukanPrima.py
Masukkan bilangan: 56
56 bukan bilangan prima
2 kali 28 = 56
```

3. Menemukan Faktor sebuah Bilangan

Kode Program *Kasus3_TentukanFaktorBilangan.py*

```
# Fungsi menerima input bilangan dan mencetak faktornya
def print_faktor(x):
    print("Faktor dari", x, "adalah:")
    for i in range(1, x+1):
        if x % i == 0:
            print(i)


# Input bilangan yang akan dicari faktornya
num = int(input("Masukkan bilangan: "))

print_faktor(num)
```

Output *Kasus3_TentukanFaktorBilangan.py*

```
(komputerteknik)
D:\Kasus3_TentukanFaktorBilangan.py
Masukkan bilangan: 9
Faktor dari 9 adalah:
1
3
9
```

```
(komputerteknik)
D:\Kasus3_TentukanFaktorBilangan.py
Masukkan bilangan: 57
Faktor dari 57 adalah:
1
3
19
57
```

4. Menyelesaikan persamaan kuadrat $ax^2 + bx + c = 0$

Kode Program *Kasus4_MemecahkanPersamaanKuadrat.py*

```
import math

# Input koefisien dari a, b, c dari ax2 + bx + c = 0
a = int(input('Masukkan a: '))
b = int(input('Masukkan b: '))
c = int(input('Masukkan c: '))

d = (b**2)-(4*a*c)  # menghitung diskriminan d

if d < 0:
    print("Persamaan ini tidak memiliki solusi real")
elif d == 0:
    x = (-b+math.sqrt(d))/(2*a)
    print("Persamaan ini hanya punya satu solusi: ", x)
else:
    x1 = (-b+math.sqrt(d))/(2*a)
    x2 = (-b-math.sqrt(d))/(2*a)
    print("Persamaan ini punya dua solusi: ", x1, " dan ", x2)
```

4. Menyelesaikan persamaan kuadrat $ax^2 + bx + c = 0$

Output *Kasus4_MemecahkanPersamaanKuadrat.py*

```
(komputerteknik) D:\Kasus4_MemecahkanPersamaanKuadrat.py
Masukkan a: 1
Masukkan b: 5
Masukkan c: 6
Persamaan ini punya dua solusi:  -2.0  dan  -3.0
```

```
(komputerteknik) D:\Kasus4_MemecahkanPersamaanKuadrat.py
Masukkan a: 1
Masukkan b: 2
Masukkan c: 1
Persamaan ini hanya punya satu solusi:  -1.0
```

```
(komputerteknik) D:\Kasus4_MemecahkanPersamaanKuadrat.py
Masukkan a: 1
Masukkan b: 3
Masukkan c: 7
Persamaan ini tidak memiliki solusi real
```

5. Menemukan Faktor Persekutuan Besar (FPB) dari dua buah bilangan

Kode Program *Kasus5_FPBduabilangan.py*

```
# mendefinisikan fungsi
def hitung_FPB(x, y):
    # memilih bilangan yang paling kecil
    if x > y:
        smaller = y
    else:
        smaller = x
    for i in range(1, smaller+1):
        if((x % i == 0) and (y % i == 0)):
            fpb = i

    return fpb


# hilangkan tanda # untuk men-test program dengan angka 96 dan 24
num1 = 96
num2 = 24

# hilangkan tanda # untuk meminta inputan angka dari user
# num1 = int(input("Enter first number: "))
# num2 = int(input("Enter second number: "))

print("FPB dari", num1, "dan", num2, " =", hitung_FPB(num1, num2))
```

5. Menemukan Faktor Persekutuan Besar (FPB) dari dua buah bilangan

Output Program *Kasus5_FPBduabilangan.py*

```
(komputerteknik) D:\Kasus5_FPBduabilangan.py
FPB dari 96 dan 24  = 24
```

```
(komputerteknik) D:\Kasus5_FPBduabilangan.py
Enter first number: 16
Enter second number: 36
FPB dari 16 dan 36  = 4
```

```
(komputerteknik) D:\Kasus5_FPBduabilangan.py
Enter first number: 45
Enter second number: 40
FPB dari 45 dan 40  = 5
```

6. Kalkulator sederhana

Kode Program *Kasus6_KalkulatorSederhana.py*

```
# fungsi penjumlahan
def add(x, y):
    return x + y

# fungsi pengurangan
def subtract(x, y):
    return x - y

# fungsi perkalian
def multiply(x, y):
    return x * y

# fungsi pembagian
def divide(x, y):
    return x / y

# menu operasi
print("Pilih Operasi.")
print("1.Jumlah")
print("2.Kurang")
print("3.Kali")
print("4.Bagi")
```

6. Kalkulator sederhana

Kode Program *Kasus6_KalkulatorSederhana.py*

```
# Meminta input dari pengguna
choice = input("Masukkan pilihan operasi (1/2/3/4): ")

num1 = int(input("Masukkan bilangan pertama: "))
num2 = int(input("Masukkan bilangan kedua: "))

if choice == '1':
    print(num1, "+", num2, "=", add(num1, num2))
elif choice == '2':
    print(num1, "-", num2, "=", subtract(num1, num2))
elif choice == '3':
    print(num1, "*", num2, "=", multiply(num1, num2))
elif choice == '4':
    print(num1, "/", num2, "=", divide(num1, num2))
else:
    print("Input salah")
```

6. Kalkulator sederhana

Output Program *Kasus6_KalkulatorSederhana.py*

```
(komputerteknik) D:\Kasus6_KalkulatorSederhana.py
Pilih Operasi.
1.Jumlah
2.Kurang
3.Kali
4.Bagi
Masukkan pilihan operasi (1/2/3/4): 1
Masukkan bilangan pertama: 5
Masukkan bilangan kedua: 5
5 + 5 = 10
```

```
(komputerteknik) D:\Kasus6_KalkulatorSederhana.py
Pilih Operasi.
1.Jumlah
2.Kurang
3.Kali
4.Bagi
Masukkan pilihan operasi (1/2/3/4): 3
Masukkan bilangan pertama: 6
Masukkan bilangan kedua: 7
6 * 7 = 42
```

7. Mencetak semua Permutasi

Kode Program *Kasus7_CetakSemuaPermutasi.py*

```
# Program Python untuk mencetak semua permutasi.
# Jumlah permutasi untuk sejumlah n anggota adalah n!
# Program ini menggunakan fungsi permutations dari paket itertools
from itertools import permutations

# Mendapatkan semua permutasi dari [1, 2, 3]
perm = permutations([1, 2, 3])

# Print semua permutasi
for i in perm:
    print(i)
```

7. Mencetak semua Permutasi

Output Program *Kasus7_CetakSemuaPermutasi.py*

```
(komputerteknik) D:\Kasus7_CetakSemuaPermutasi.py
(1, 2, 3)
(1, 3, 2)
(2, 1, 3)
(2, 3, 1)
(3, 1, 2)
(3, 2, 1)
```

8. Mencetak semua Permutasi sejumlah n anggota

Kode Program *Kasus8_CetakPermutasiTertentu.py*

```
# Program Python untuk mencetak semua permutasi dgn panjang ditentukan.
# Jumlah permutasi untuk sejumlah n anggota adalah n!
# Program ini menggunakan fungsi permutations dari paket itertools
from itertools import permutations

# Mendapatkan semua permutasi dari [1, 2, 3]
perm = permutations([1, 2, 3], 2)

# Print semua permutasi
for i in perm:
    print(i)
```

8. Mencetak semua Permutasi sejumlah n anggota

Output Program *Kasus8_CetakPermutasiTertentu.py*

```
(komputerteknik) D:\Kasus8_CetakPermutasiTertentu.py
(1, 2)
(1, 3)
(2, 1)
(2, 3)
(3, 1)
(3, 2)
```

9. Mencetak semua Kombinasi dengan panjang r tertentu

Kode Program *Kasus9_KombinasiDgnPanjangTertentu.py*

```
# Program Python untuk mencetak semua
# kombinasi dengan panjang r tertentu
from itertools import combinations

# Mendapatkan semua kombinasi dari [1, 2, 3]
# dengan panjang 2
comb = combinations([1, 2, 3], 2)

# Print semua kombinasi
for i in comb:
    print(i)
```

9. Mencetak semua Kombinasi dengan panjang r tertentu

Output Program *Kasus9_KombinasiDgnPanjangTertentu.py*

```
(komputerteknik) D:\Kasus9_KombinasiDgnPanjangTertentu.py
(1, 2)
(1, 3)
(2, 3)
```

10. Mencetak semua Kombinasi dengan elemen sama

Kode Program *Kasus10_KombinasiDgElemenSama.py*

```
# Program Python untuk menampilkan semua kombinasi
# dengan mengizinkan kombinasi anggota yang sama
from itertools import combinations_with_replacement

# Mendapatkan kombinasi dari [1, 1, 3]
# dan panjang 2
comb = combinations_with_replacement([1, 2, 3], 2)

# Menampilkan kombinasi
for i in comb:
    print(i)
```

10. Mencetak semua Kombinasi dengan elemen sama

Output Program *Kasus10_KombinasiDgElemenSama.py*

```
(komputerteknik) D:\Kasus10_KombinasiDgElemenSama.py
(1, 1)
(1, 2)
(1, 3)
(2, 2)
(2, 3)
(3, 3)
```

11. Menghitung Zakat Penghasilan

Kode Program *Kasus11_MenghitungZakatPenghasilan.py*

```
# Menghitung zakat penghasilan
# dengan nisab harga emas per gram.
nama = []
gaji = []
emas = []
zakat = []
pertahun = []
perbulan = []
nisab = []
print('+————————————————-+')
print('| Penghitung Zakat Penghasilan |')
print('| menurut pendapatan kasar (brutto) |')
print('| |')
print('+————————————————-+')
data = int(input('Masukan banyak data : '))
print('== == == == == == == == == == == == == == == == == == == == == ')

for i in range(data):
    a = input('Masukan nama : ')
    nama.append(a)
    b = int(input('Masukan harga emas per gram saat ini: '))
    emas.append(b)
    c = int(input('Masukkan penghasilan Anda per bulan : '))
    gaji.append(c)
    print('')
```

11. Menghitung Zakat Penghasilan

Kode Program *Kasus11_MenghitungZakatPenghasilan.py*

```
for i in range(data):
    d = 12 * gaji[i]
    pertahun.append(d)
    e = 0.025 * pertahun[i]
    zakat.append(e)
    f = 85 * emas[i]
    nisab.append(f)
    g = zakat[i] / 12
    perbulan.append(g)

for i in range(data):
    print('')
    print('————————————-')
    print(' Zakat Penghasilan (Brutto)')
    print('————————————-')
    print('Nama :', nama[i])
    print('Harga 1 gram emas :', 'Rp.', emas[i])
    print('Penghasilan per bulan :', 'Rp.', gaji[i])
    print('Penghasilan per tahun :', 'Rp.', pertahun[i])
    print('Harga nishab (85 gram emas) :', 'Rp.', nisab[i])
    print('Zakat penghasilan :', '2.5% x', pertahun[i], '=', 'Rp.', zakat[i])
    if pertahun[i] >= nisab[i]:
        print('Keterangan : WAJIB Zakat Rp.', zakat[i], '/tahun')
        print(' atau Rp.', perbulan[i], '/bulan')
    print('')
    if pertahun[i] <= nisab[i]:
        print('Keterangan : Anda belum termasuk Wajib Zakat')
```

11. Menghitung Zakat Penghasilan

Output Program *Kasus11_MenghitungZakatPenghasilan.py*

```
(komputerteknik) D:\Kasus11_MenghitungZakatPenghasilan.py
+-----------------+
| Penghitung Zakat Penghasilan |
| menurut pendapatan kasar (brutto) |
| |
+-----------------+
Masukan banyak data : 1
== == == == == == == == == == == == == == == == == == == == ==
Masukan nama : ratna
Masukan harga emas per gram saat ini: 911000
Masukkan penghasilan Anda per bulan : 7000000



-------------
 Zakat Penghasilan (Brutto)
-------------
Nama : ratna
Harga 1 gram emas : Rp. 911000
Penghasilan per bulan : Rp. 7000000
Penghasilan per tahun : Rp. 84000000
Harga nishab (85 gram emas) : Rp. 77435000
Zakat penghasilan : 2.5% x 84000000 = Rp. 2100000.0
Keterangan : WAJIB Zakat Rp. 2100000.0 /tahun
 atau Rp. 175000.0 /bulan
```

12. Menentukan Nilai Akhir Semester

Kode Program *Kasus12_MenentukanNilaiSemester.py*

```
# Program menentukan nilai akhir semester
# Baik dalam angka maupun huruf (4 grade)

# Deklarasi Fungsi Operator
def fungsi_total_nilai(var_harian, var_uts, var_uas):
    var_harian = int(var_harian) * 0.3
    var_uts = int(var_uts) * 0.3
    var_uas = int(var_uas) * 0.4

    var_total = var_harian + var_uts + var_uas
    return var_total


# Deklarasi Fungsi Percabangan
def fungsi_percabangan(var_nilai):
    var_huruf = ""
    if (var_nilai >= 0 and var_nilai < 20):
        var_huruf = "E"
    elif (var_nilai >= 20 and var_nilai < 40):
        var_huruf = "D"
    elif (var_nilai >= 40 and var_nilai < 60):
        var_huruf = "C"
    elif (var_nilai >= 60 and var_nilai < 80):
        var_huruf = "B"
    elif (var_nilai >= 80 and var_nilai < 100):
        var_huruf = "A"
    return var_huruf
```

12. Menentukan Nilai Akhir Semester

Kode Program *Kasus12_MenentukanNilaiSemester.py*

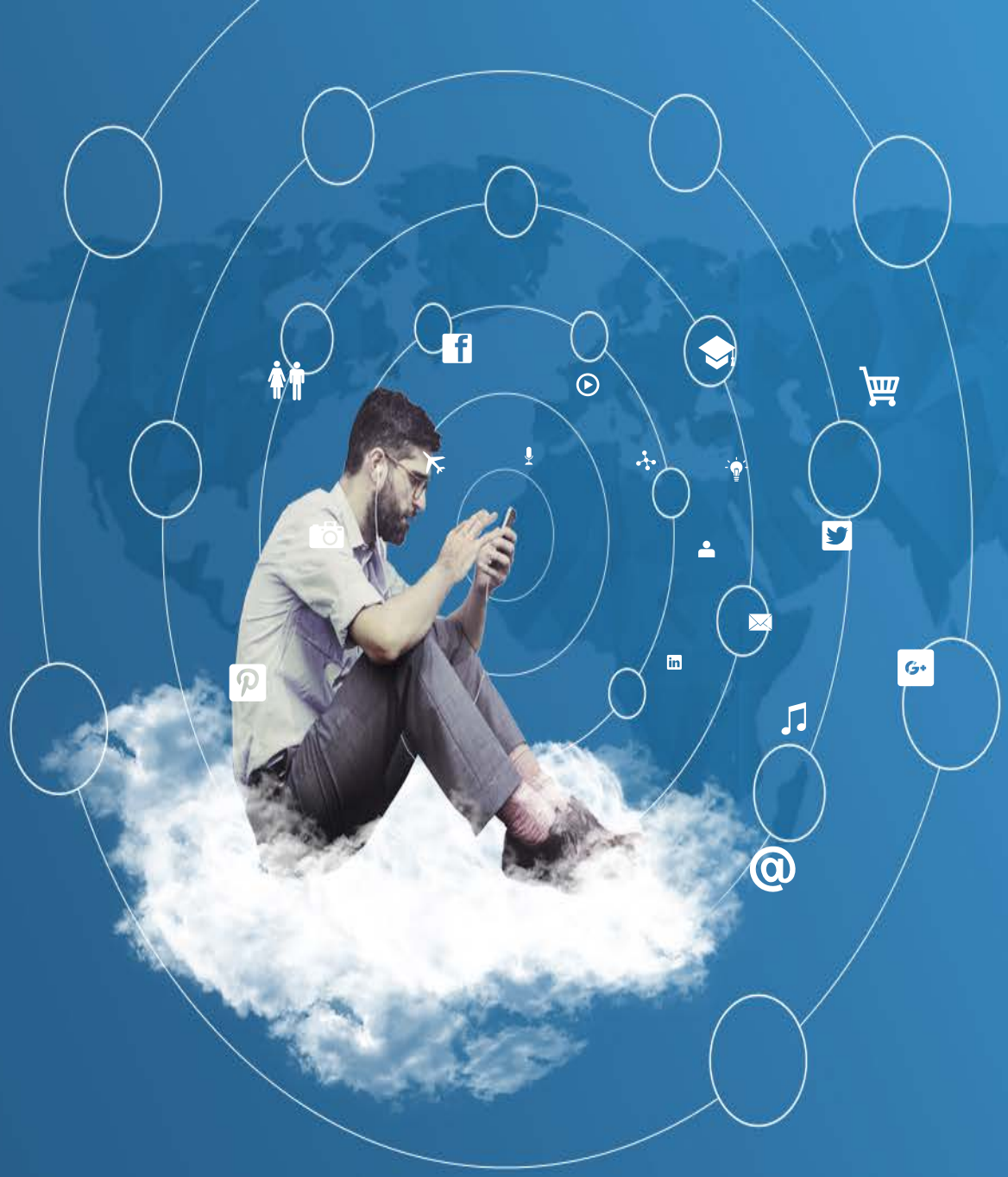
```
# Deklarasi Fungsi Perulangan
def fungsi_perulangan():
    var_hasil_perulangan = 0
    for i in range(1, 3):
        print("——-Nilai Ke ", i, "——-")
        var_harian = input("Nilai Harian : ")
        var_uts = input("Nilai UTS : ")
        var_uas = input("Nilai UAS : ")

        # Pemanggilan fungsi Penjumlahan
        var_hasil_perulangan += (int(fungsi_total_nilai(var_harian, var_uts, var_uas)))

    return var_hasil_perulangan / i


# Membuat eksekusi program dimulai dari main ini
if __name__ == "__main__":
    # Pemanggilan fungsi perulangan
    var_total = fungsi_perulangan()

    print("——-Total Nilai ——-")
    print("Total nilai yang didapat : ", var_total)

    # Pemanggilan Fungsi Percabangan
    print("Total Nilai Dalam Huruf : ", fungsi_percabangan(var_total))
```

12. Menentukan Nilai Akhir Semester

Output Program *Kasus12_MenentukanNilaiSemester.py*

```
(komputerteknik) D:\Kasus12_MenentukanNilaiSemester.py
---Nilai Ke  1 ---
Nilai Harian : 70
Nilai UTS : 70
Nilai UAS : 70
---Nilai Ke  2 ---
Nilai Harian : 90
Nilai UTS : 90
Nilai UAS : 90
---Total Nilai ---
Total nilai yang didapat :  80.0
Total Nilai Dalam Huruf :  A
```

Thank You



Programming for Science

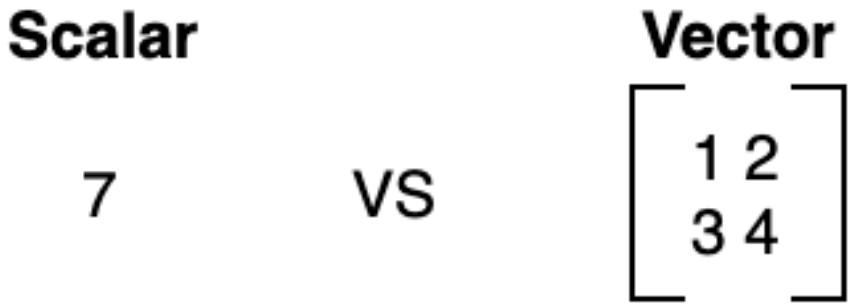
Manipulasi Matriks dengan Python

Menambahkan Paket Library

- Sebelum kita memprogram Python untuk keperluan operasi matriks, maka kita harus memasukkan paket library **Numpy** ke dalam *virtual environment (venv)* python yang ada.
- Dalam hal ini *virtual environment* saya adalah 'komputerteknik' maka sesuaikan dengan *venv* anda yang biasa dipakai pada perkuliahan ini.
- Pada prompt anaconda anda lakukan langkah[2] sbb:
  1. Pindahkan *venv* aktif *default* (base) ke venv anda dengan perintah: `(base)>conda activate [nama-venv]`
  2. Install paket library **Numpy** ketika *venv* anda telah aktif dengan perintah: `(nama_venv)>conda install numpy`

Apa Itu Matriks

- Di sini kita akan membandingkan matriks dengan skalar.



- Matriks adalah array 2D, di mana setiap elemen dalam array diatas memiliki 2 indeks. Misalnya, [[1, 2], [3, 4]] adalah matriks, dan indeks 1 dalam python di identifikasi dengan (0,0).
- Ketika berbicara tentang bentuk matriks, maka indeks nya adalah "baris x kolom".

Apa Itu Matriks

- Kita dapat mengatakan bahwa matriks-1 di samping ini memiliki bentuk 2x2, dan matriks-2 memiliki bentuk 3x2
- Sebaliknya, skalar hanyalah angka, seperti angka 5.

Matriks-1:

```
[[1, 2],
 [3, 4]]
```

Matriks-2:

```
[[10, 20],
 [11, 21],
 [12, 22]]
```

01. Mencetak indeks dari matriks

Kode Program *01_MencetakIndeksMatriks.py*

```
import numpy as np

# Membuat matrik A
A = np.array([[1, 2],
              [3, 4]])

# Mengambil/mencetak indeks 0,0 --> yaitu 1
K = np.array(A)[0, 0]
print(K)

# Mengambil/mencetak indeks 1,0 --> yaitu 3
print(np.array(A)[1, 0])
```

01. Mencetak indeks dari matriks

Output Program *01_MencetakIndeksMatriks.py*

```
(komputerteknik) D:\01_MencetakIndeksMatriks.py
1
3
```

02. Perkalian skalar dengan matriks

- Bentuk sederhana dari perkalian matriks disebut perkalian skalar, mengalikan skalar dengan matriks.

$$7 \times \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} = \begin{bmatrix} 1x7 & 2x7 \\ 3x7 & 4x7 \end{bmatrix} = \begin{bmatrix} 7 & 14 \\ 21 & 28 \end{bmatrix}$$

- Perkalian skalar umumnya mudah. Setiap nilai dalam matriks input dikalikan dengan skalar, dan output memiliki bentuk yang sama dengan matriks input.
- Mari kita lakukan contoh di atas tetapi dengan Python's Numpy.

02. Perkalian skalar dengan matriks

Kode Program *02_PerkalianSkalarDgMatriks.py*

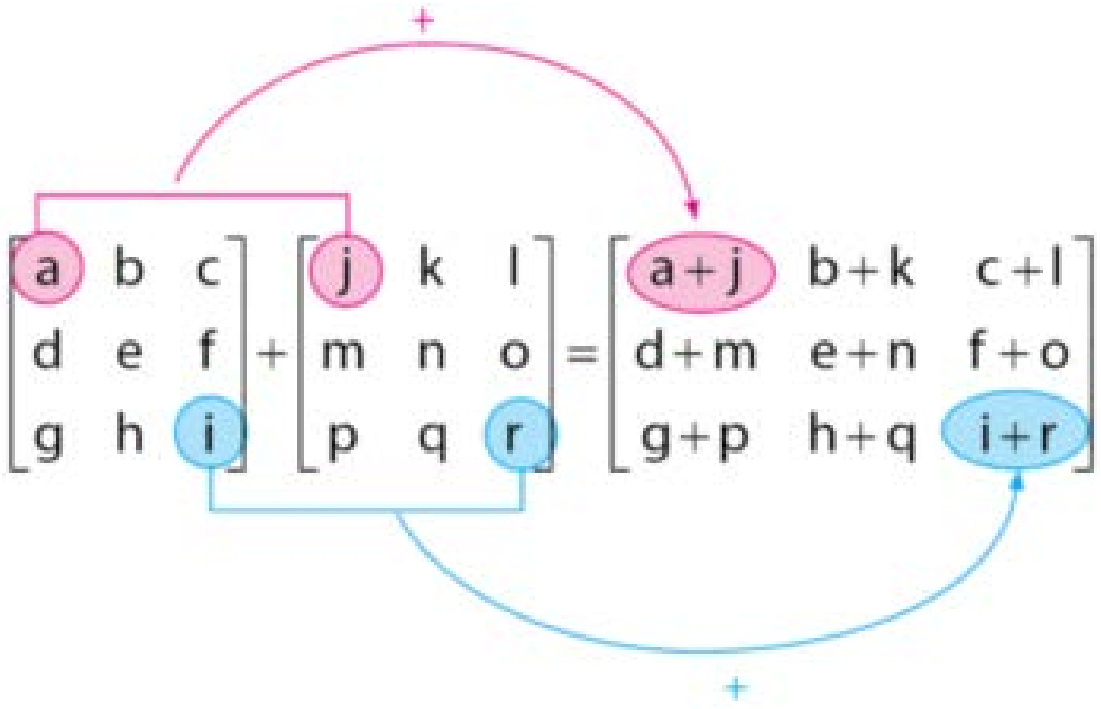
```
import numpy as np

a = 7
B = np.array([[1, 2],
              [3, 4]])

Array = np.dot(a, B)
print(Array)
```

Output Program *02_PerkalianSkalarDgMatriks.py*

```
(komputerteknik) D:\02_PerkalianSkalarDgMatriks.py
[[ 7 14]
 [21 28]]
```

03. Penambahan matriks

- Contoh cara melakukan operasi penjumlahan pada matriks:



- Contoh :

$$\begin{bmatrix} 4 & -1 \\ 3 & 2 \end{bmatrix} + \begin{bmatrix} 2 & 1 \\ -4 & 0 \end{bmatrix} = \begin{bmatrix} 4+2 & -1+1 \\ 3+(-4) & 2+0 \end{bmatrix} = \begin{bmatrix} 6 & 0 \\ -1 & 2 \end{bmatrix}$$

03. Penambahan matriks

Kode Program *03_PenambahanMatriks.py*

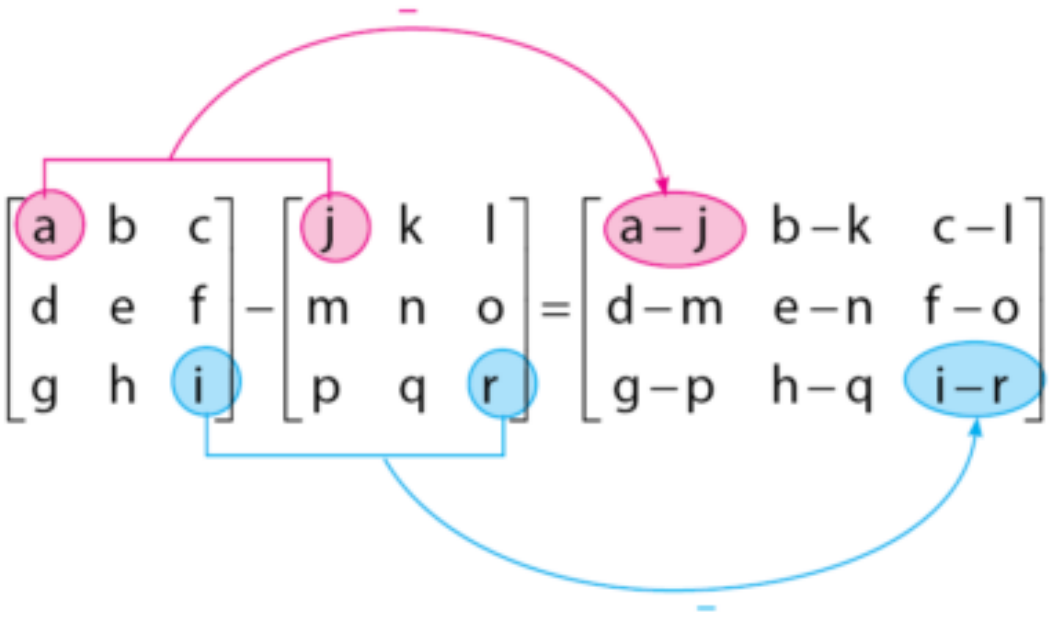
```
import numpy as np

A = np.array([[4, -1],
              [3, 2]])
B = np.array([[2, 1],
              [-4, 0]])

Array = np.add(A, B)
print(Array)
```

Output Program *03_PenambahanMatriks.py*

```
(komputerteknik) D:\03_PenambahanMatriks.py
[[ 6  0]
 [-1  2]]
```

04. Pengurangan matriks

- Contoh cara melakukan operasi penjumlahan pada matriks:



- Contoh :

$$\begin{pmatrix} 4 & -1 \\ 3 & 2 \end{pmatrix} - \begin{pmatrix} 2 & 1 \\ -3 & 0 \end{pmatrix} = \begin{pmatrix} 4-2 & -1-1 \\ 3-(-3) & 2-0 \end{pmatrix} = \begin{pmatrix} 2 & -2 \\ 6 & 2 \end{pmatrix}$$

04. Pengurangan matriks

Kode Program *04_PenguranganMatriks.py*

```
import numpy as np

A = np.array([[4, -1],
              [3, 2]])
B = np.array([[2, 1],
              [-3, 0]])

Array = np.subtract(A, B)
print(Array)
```

Output Program *04_PenguranganMatriks.py*

```
(komputerteknik) D:\04_PenguranganMatriks.py
[[ 2 -2]
 [ 6  2]]
```

05_a. Perkalian matriks

- Contoh cara melakukan operasi perkalian pada matriks:

$$\mathbf{A}\cdot\mathbf{B}=\begin{pmatrix} a & b \\ c & d \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix}=\begin{pmatrix} ax+by \\ cx+dy \end{pmatrix}$$

$$\mathbf{A}\cdot\mathbf{B}=\begin{pmatrix} a & b \\ c & d \end{pmatrix}\begin{pmatrix} k & l \\ m & n \end{pmatrix}=\begin{pmatrix} ak+bm & al+bn \\ ck+dm & cl+dn \end{pmatrix}$$

- Contoh :

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \times \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix} = \begin{bmatrix} 1\times5+2\times7 & 1\times6+2\times8 \\ 3\times5+4\times7 & 3\times6+4\times8 \end{bmatrix} = \begin{bmatrix} 5+14 & 6+16 \\ 15+28 & 18+32 \end{bmatrix} = \begin{bmatrix} 19 & 22 \\ 43 & 50 \end{bmatrix}$$

05_a. Perkalian matriks

Kode Program *05_a_PerkalianMatriks.py*

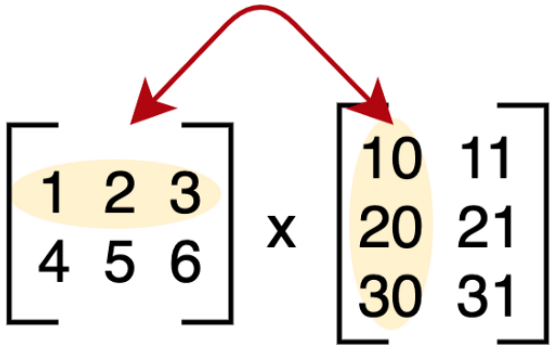
```
import numpy as np

A = np.array([[1, 2],
              [3, 4]])
B = np.array([[5, 6],
              [7, 8]])

Array = np.dot(A, B)
print(Array)
```

Output Program *05_a_PerkalianMatriks.py*

```
(komputerteknik) D:\05_PerkalianMatriks.py
[[19 22]
 [43 50]]
```

05_b. Perkalian matriks

- Contoh cara melakukan operasi perkalian pada matriks:



$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \times \begin{bmatrix} 10 & 11 \\ 20 & 21 \\ 30 & 31 \end{bmatrix}$$

$$= \begin{bmatrix} 1x10 + 2x20 + 3x30 & 1x11 + 2x21 + 3x31 \\ 4x10 + 5x20 + 6x30 & 4x11 + 5x21 + 6x31 \end{bmatrix}$$

$$= \begin{bmatrix} 10+40+90 & 11+42+93 \\ 40+100+180 & 44+105+186 \end{bmatrix} = \begin{bmatrix} 140 & 146 \\ 320 & 335 \end{bmatrix}$$

05_b. Perkalian matriks

Kode Program *05_b_PerkalianMatriks.py*

```
import numpy as np

A = np.array([[1, 2, 3],
              [4, 5, 6]])
B = np.array([[10, 11],
              [20, 21],
              [30, 31]])

Array = np.dot(A, B)
print(Array)
```

Output Program *05_b_PerkalianMatriks.py*

```
(komputerteknik) D:\05_b_PerkalianMatriks.py
[[140 146]
 [320 335]]
```

05_c. Perkalian matriks

- Contoh cara melakukan operasi perkalian pada matriks:

$$\begin{bmatrix} 1 & 2 & 3 & 4 & 5 \end{bmatrix} \times \begin{bmatrix} 10 \\ 20 \\ 30 \\ 40 \\ 50 \end{bmatrix}$$

$$= \begin{bmatrix} 1x10 + 2x20 + 3x30 + 4x40 + 5x50 \end{bmatrix}$$

$$= \begin{bmatrix} 10 + 40 + 90 + 160 + 250 \end{bmatrix}$$

$$= \begin{bmatrix} 550 \end{bmatrix}$$

05_c. Perkalian matriks

Kode Program *05_c_PerkalianMatriks.py*

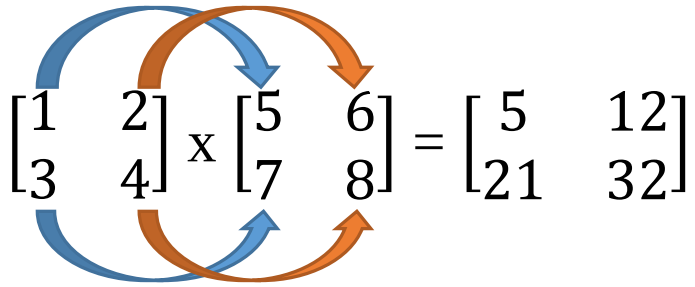
```
import numpy as np

A = np.array([[1, 2, 3, 4, 5]])
B = np.array([[10],
              [20],
              [30],
              [40],
              [50]])

Array = np.dot(A, B)
print(Array)
```

Output Program *05_c_PerkalianMatriks.py*

```
(komputerteknik) D:\05_c_PerkalianMatriks.py
[[550]]
```

05_d. Perkalian matriks elemen

- Fungsi *numpy.multiply* ini digunakan untuk melakukan perkalian matriks elemen.
- Syaratnya dua matrik yang akan di *multiply* harus memiliki ordo (dimensi) yang sama.
- Contoh cara melakukan operasi perkalian pada matriks elemen-elemen nya:



05_d. Perkalian matriks elemen

Kode Program *05_d_PerkalianMatriksElemen.py*

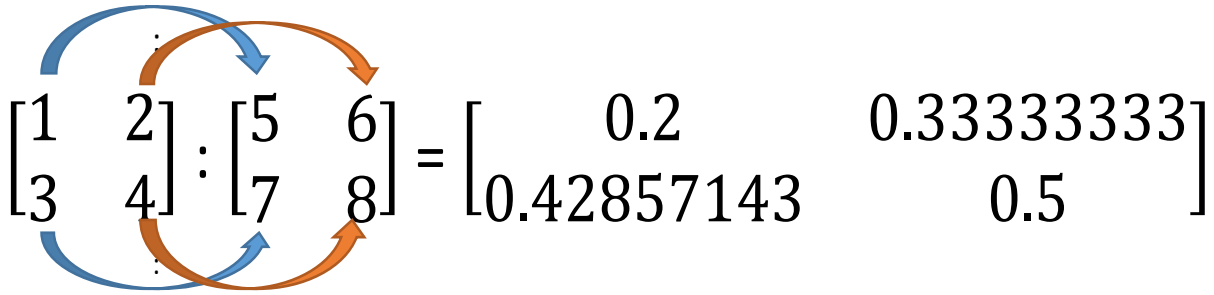
```
import numpy as np

A = np.array([[1, 2],
              [3, 4]])
B = np.array([[5, 6],
              [7, 8]])

Array = np.multiply(A, B)
print(Array)
```

Output Program *05_d_PerkalianMatriksElemen.py*

```
(komputerteknik) D:\ 05_d_PerkalianMatriksElemen.py
[[ 5 12]
 [21 32]]
```

06. Pembagian matriks elemen

- Fungsi *numpy.divide* ini digunakan untuk melakukan pembagian matriks elemen.
- Syaratnya dua matrik yang akan di *divide* harus memiliki ordo (dimensi) yang sama.
- Contoh cara melakukan operasi perkalian pada matriks elemen-elemen nya:



06. Pembagian matriks elemen

Kode Program *06_PembagianMatriksElemen.py*

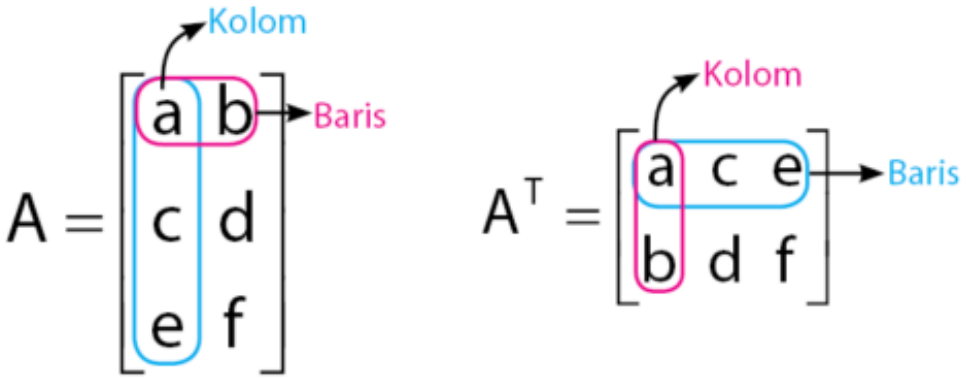
```
import numpy as np

A = np.array([[1, 2],
              [3, 4]])
B = np.array([[5, 6],
              [7, 8]])

Array = np.divide(A, B)
print(Array)
```

Output Program *06_PembagianMatriksElemen.py*

```
(komputerteknik) D:\06_PembagianMatriksElemen.py
[[0.2        0.33333333]
 [0.42857143 0.5       ]]
```

07. Matriks Tranpose

- Transpose matriks A disimbolkan dengan A^T. Matriks transpose A^T adalah matriks yang diperoleh dengan cara menukar elemen pada baris menjedi elemen pada kolom.
- Contoh cara melakukan operasi tranpose pada matriks elemen-elemen nya:



$$\mathbf{A} = \begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix} \Rightarrow \mathbf{A}^T = \begin{pmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{pmatrix}$$

07. Matriks Tranpose

Kode Program *07_MatriksTranpose.py*

```
import numpy as np

A = np.array([[1, 2],
              [3, 4],
              [5, 6]])

Array = A.T
print(Array)
```

Output Program *07_MatriksTranpose.py*

```
(komputerteknik) D:\07_MatriksTranpose.py
[[1 3 5]
 [2 4 6]]
```

08. Menyelesaikan Persamaan Linear

- Diketahui persamaan Aljabar linear sebagai berikut :

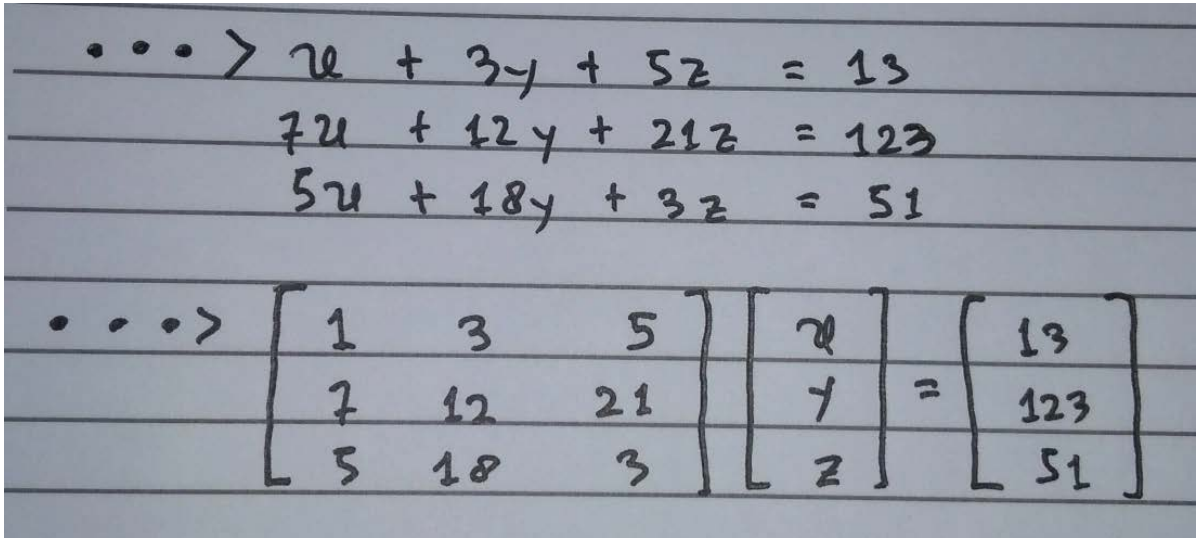
  $x + 3y + 5z = 13$

  $7x + 12y + 21z = 123$

  $5x + 18y + 3z = 51$

- Tentukanlah nilai x, y, dan z dari persamaan tersebut! Gunakan Python untuk menyelesaikan persamaan tersebut!

- Rubah persamaan linear ke dalam bentuk matriks.



08. Menyelesaikan Persamaan Linear

Kode Program *08_AljabarLinearSolving.py*

```
import numpy as np

A = np.array([[1, 3, 5],
              [7, 12, 21],
              [5, 18, 3]])

B = np.array([[13],
              [123],
              [51]])

Array = np.linalg.solve(A, B)
print(Array)
```

Solusi untuk persamaan tersebut yaitu x = 23.625, y = -3.75, dan z = 0.125.

Output Program *08_AljabarLinearSolving.py*

```
(komputerteknik) D:\08_AljabarLinearSolving.py
[[23.625]
 [-3.75 ]
 [ 0.125]]
```

09. Menyelesaikan Persamaan Linear

- Diketahui persamaan Aljabar linear dengan 4 variabel sebagai berikut :

$$\begin{aligned} x + y + z + w &= 13 \\ 2x + 3y - w &= 1 \\ -3x + 4y + z + 2w &= 10 \\ x + 2y - z + w &= 1 \end{aligned}$$

- Rubah ke dalam bentuk matriks :

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 3 & 0 & -1 \\ -3 & 4 & 1 & 2 \\ 1 & 2 & -1 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} = \begin{bmatrix} 13 \\ 1 \\ 10 \\ 1 \end{bmatrix}$$

09. Menyelesaikan Persamaan Linear

Kode Program *08_AljabarLinearSolving.py*

```
import numpy as np

A = np.array([[1, 1, 1, 1],
              [2, 3, 0, -1],
              [-3, 4, 1, 2],
              [1, 2, -1, 1]])

B = np.array([[13],
              [1],
              [10],
              [1]])

Array = np.linalg.solve(A, B)
print(Array)
```

Solusi untuk persamaan tersebut yaitu x = 2.11111111, y = 0.37037037, z = 6.18518519, dan w = 4.33333333. Output Program *08_AljabarLinearSolving.py*

```
(komputerteknik) D:\09_AljabarLinearSolving.py
[[2.11111111]
 [0.37037037]
 [6.18518519]
 [4.33333333]]
```

Thank You