



Belajar Python (Gratis!)

Python adalah bahasa pemrograman high-level yang sangat *powerful*, sintaksnya sederhana dan mudah dipelajari, juga memiliki performa yang bagus. Python memiliki komunitas yang besar, bahasa ini dipakai di berbagai platform diantaranya: web, data science, infrastructure tooling, dan lainnya.

E-book Dasar Pemrograman Python ini cocok untuk pembaca yang ingin mempelajari pemrograman python dalam kurun waktu yang relatif cepat, dan gratis. Konten pembelajaran pada ebook ini disajikan secara ringkas tidak bertele-tele tapi tetap mencakup point penting yang harus dipelajari.

Selain topik fundamental python programming, nantinya akan disediakan juga pembahasan *advance* lainnya, **stay tuned!**

Versi e-book: **v1.0.0-beta1.20230430**, dan versi **Python 3.10.6**.

E-book ini aktif dalam pengembangan, kami akan tambah terus konten-kontennya. Silakan cek di [Github repo](#) kami mengenai progress development e-book.

Download Ebook File (pdf)

Ebook ini bisa di-download dalam bentuk file, silakan gunakan link berikut:

Source Code Praktik

Source code contoh program bisa diunduh di github.com/novalagung/dasarpemrogramanpython-example. Dianjurkan untuk sekedar tidak copy-paste dari source code dalam proses belajar, usahakan tulis sendiri kode program agar cepat terbiasa dengan bahasa Rust.

Kontribusi

Ebook ini merupakan project open source, teruntuk siapapun yang ingin berkontribusi silakan langsung saja cek github.com/novalagung/dasarpemrogramanpython. Cek juga [halaman kontributor](#) untuk melihat list kontributor.

Lisensi dan Status FOSSA

Ebook Dasar Pemrograman Rust gratis untuk disebarluaskan secara bebas, baik untuk komersil maupun tidak, dengan catatan harus disertakan credit sumber aslinya (yaitu Dasar Pemrograman Rust atau novalagung) dan tidak mengubah lisensi aslinya (yaitu CC BY-SA 4.0). Lebih jelasnya silakan cek [halaman lisensi dan distribusi konten](#).

FOSSA Status

Author & Maintainer

Ebook ini dibuat oleh Noval Agung Prayogo. Untuk pertanyaan, kritik, dan saran, silakan drop email ke [\[email protected\]](#).



Author & Contributors

Ebook Dasar Pemrograman Python adalah project open source. Siapapun bebas untuk berkontribusi di sini, bisa dalam bentuk perbaikan typo, update kalimat, maupun submit tulisan baru.

Bagi teman-teman yang berminat untuk berkontribusi, silakan fork github.com/novalagung/dasarpemrogramanpython, kemudian langsung saja cek/buat issue kemudian submit relevan pull request untuk issue tersebut 😊.

Original Author

E-book ini di-inisialisasi oleh Noval Agung Prayogo.

Contributors

Berikut merupakan hall of fame kontributor yang sudah berbaik hati menyisihkan waktunya untuk membantu pengembangan e-book ini.

1. ... anda :-)



Lisensi & Distribusi Konten

Ebook Dasar Pemrograman Python gratis untuk disebarluaskan secara bebas, dengan catatan sesuai dengan aturan lisensi CC BY-SA 4.0 yang kurang lebih sebagai berikut:

- Diperbolehkan menyebar, mencetak, dan menduplikasi material dalam konten ini ke siapapun.
- Diperbolehkan memodifikasi, mengubah, atau membuat konten baru menggunakan material yang ada dalam ebook ini untuk keperluan komersil maupun tidak.

Dengan catatan:

- Harus ada credit sumber aslinya, yaitu Dasar Pemrograman Python atau novalagung
- Tidak mengubah lisensi aslinya, yaitu CC BY-SA 4.0
- Tidak ditambahi restrictions baru
- Lebih jelasnya silakan cek <https://creativecommons.org/licenses/by-sa/4.0/>.

FOSSA Status

Instalasi Python

Ada banyak cara yang bisa dipilih untuk instalasi Python, silakan pilih sesuai preferensi dan kebutuhan.

Instalasi Python

🕒 Instalasi di Windows

- Via [Microsoft Store Package](#)
- Via [Official Python installer](#)
- Via [Chocolatey package manager](#)
- Via [Windows Subsystem for Linux \(WSL\)](#)

🕒 Instalasi di MacOS

- Via [Homebrew](#)
- Via [Official Python installer](#)

🕒 Instalasi di OS lainnya

- Via package manager masing-masing sistem operasi

🕒 Instalasi via source code

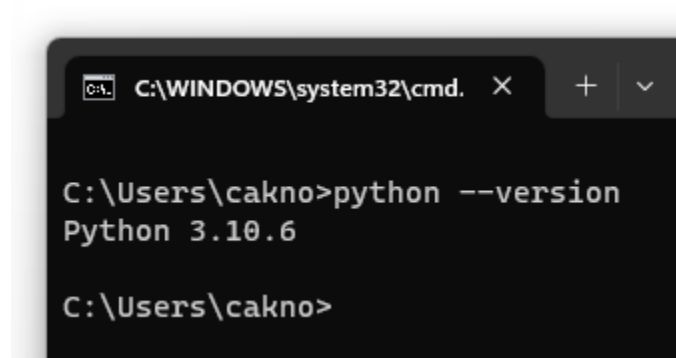
- Tarball source code bisa diunduh di [situs official Python](#)

● Instalasi via Anaconda

- File installer bisa diunduh di [situs official Anaconda](#)

Konfigurasi path Python

1. Pastikan untuk mendaftarkan path dimana Python ter-install ke OS environment variable, agar nantinya mudah dalam pemanggilan binary `python`.
2. Jika diperlukan, set juga variabel `PYTHONHOME` yang mengarah ke base folder dimana Python terinstall. Biasanya editor akan mengacu ke environment variabel ini untuk mencari dimana path Python berada.
3. Kemudian, jalankan command `python --version` untuk memastikan binary sudah terdaftar di `$PATH` variable.



```
C:\WINDOWS\system32\cmd. X + v
C:\Users\cakno>python --version
Python 3.10.6
C:\Users\cakno>
```



Python Editor & Plugin

Editor/IDE

Ada cukup banyak pilihan editor dan IDE untuk development menggunakan Python, diantaranya:

- [Eclipse](#), dengan tambahan plugin [PyDev](#)
- [GNU Emacs](#)
- [JetBrains PyCharm](#)
- [Spyder](#)
- [Sublime Text](#), dengan tambahan package [Python](#)
- [Vim](#)
- [Visual Studio](#)
- [Visual Studio Code \(VSCode\)](#), dengan tambahan extension [Python](#) dan [Jupyter](#)

Selain list di atas, ada juga editor lainnya yang bisa digunakan, contohnya seperti:

- [Python standard shell \(REPL\)](#)
- [Jupyter](#)

Preferensi editor penulis

Penulis menggunakan editor [Visual Studio Code](#) dengan tambahan:

- Extension [Python](#), untuk mendapatkan benefit API doc, autocompletion,

linting, run feature, dan lainnya.

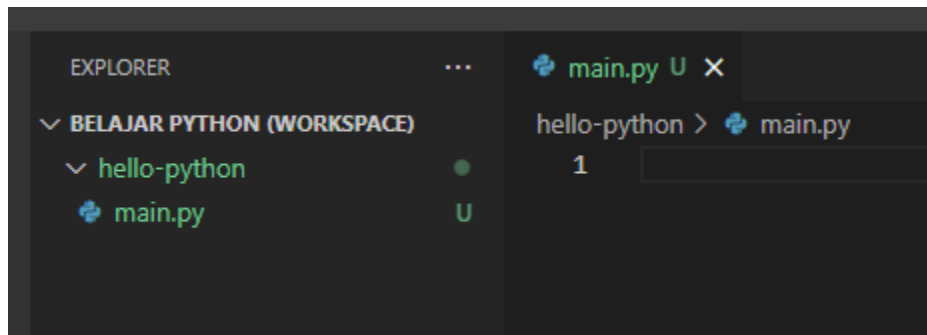
- Extension **Jupyter**, untuk interactive run program via editor.

A.1. Program Pertama → Hello Python

Bahasa pemrograman Python sangat sederhana dan mudah untuk dipelajari. Pada chapter ini kita akan langsung mempraktikannya dengan membuat program hello world.

A.1.1. Program Hello Python

Siapkan sebuah folder dengan isi satu file program Python bernama `main.py`.



Pada file `main.py`, tuliskan kode berikut:

```
print("hello python")
```

Run program menggunakan command berikut:

```
# python <nama_file_program>
```

A screenshot of a code editor interface. On the left, the 'EXPLORER' panel shows a workspace named 'BELAJAR PYTHON (WORKSPACE)' containing a folder 'hello-python' with a file 'main.py'. The main editor area shows the code in 'main.py':

```
hello-python > main.py
1 print("hello python")
```

At the bottom, the 'TERMINAL' panel shows the command 'python main.py' being executed, resulting in the output 'hello python'.

Selamat, secara official sekarang anda adalah programmer Python! 🎉 Mudah bukan!?

A.1.2. Penjelasan program

Folder `hello-python` bisa disebut dengan folder **project**, dimana isinya adalah file-file program Python berekstensi `.py`.

File `main.py` adalah file program python. Nama file program bisa apa saja, tapi umumnya pada pemrograman Python, file program utama bernama `main.py`.

Command `python <nama_file_program>` digunakan untuk menjalankan program. Cukup ganti `<nama_file_program>` dengan nama file program (yang pada contoh ini adalah `main.py`) maka kode program di dalam file tersebut akan di-run oleh Python interpreter.

Statement `print("<pesan_text>")` adalah penerapan dari salah satu fungsi *built-in* yang ada dalam Python stdlib (standard library), yaitu fungsi bernama `print()` yang kegunaannya adalah untuk menampilkan pesan string (yang disiapkan pada argument pemanggilan fungsi `print()`) yang pesan tersebut akan muncul ke layar output stdout (pada contoh ini adalah terminal milik

editor penulis).

- Lebih detailnya mengenai fungsi dibahas pada chapter *Fungsi*
- Lebih detailnya mengenai Python standard library (stdlib) dibahas terpisah pada chapter *Python standard library (stdlib)*

Catatan chapter

● Source code praktik

```
github.com/novalagung/dasarpemrogramanpython-example/./hello-python
```

● Referensi

- https://www.learnpython.org/en/Hello,_World!
- <https://docs.python.org/3/library/functions.html>



A.2. Run Python di VSCode

Chapter ini membahas tentang pilihan opsi cara run program Python di Visual Studio Code.


A.2.1. Cara run program Python di VSCode

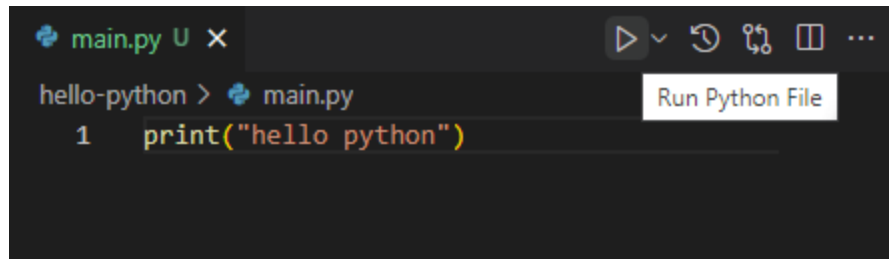
● Menggunakan command `python`

Command ini sudah kita terapkan pada chapter [Program Pertama → Hello Python](#), cara penggunaannya cukup mudah, tinggal jalankan saja command di terminal.

```
# python <nama_file_program>  
python main.py
```

● Menggunakan tombol run

Cara run program ini lebih praktis karena tinggal klik-klik saja. Di toolbar VSCode sebelah kanan atas ada tombol , gunakan tombol tersebut untuk menjalankan program.

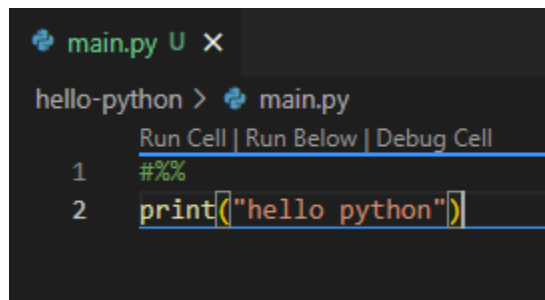


```
main.py U x
hello-python > main.py
1 print("hello python")
```

Run Python File

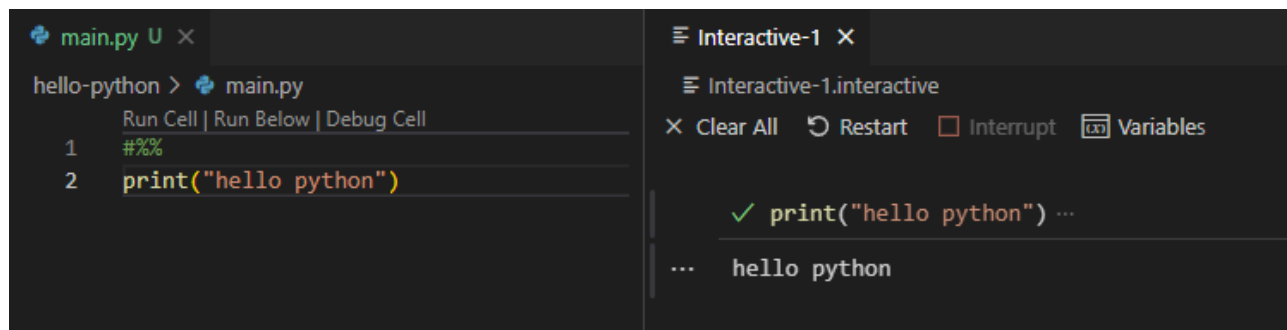
● Menggunakan jupyter code cells

Untuk menerapkan cara ini, tambahkan kode `# %%` pada baris di atas statement `print("hello python")`, dengan ini maka blok kode dianggap sebagai `code cell`.



```
main.py U x
hello-python > main.py
Run Cell | Run Below | Debug Cell
1 #%%
2 print("hello python")
```

Setelah itu, muncul tombol `Run Cell`, klik untuk run program.



```
main.py U x
hello-python > main.py
Run Cell | Run Below | Debug Cell
1 #%%
2 print("hello python")
```

Interactive-1 X

Interactive-1.interactive

Clear All Restart Interrupt Variables

✓ print("hello python") ...

... hello python

Catatan chapter

● Chapter relevan lainnya

- Program Pertama → Hello Python

● Referensi

- <https://code.visualstudio.com/docs/python/python-tutorial>
- <https://code.visualstudio.com/docs/datascience/jupyter-notebooks>
- <https://docs.python.org/3/using/cmdline.html>

A.3. Komentar

Komentar adalah sebuah statement yang tidak akan dijalankan oleh interpreter. Biasanya digunakan untuk menambahkan keterangan atau men-disable statements agar tidak dieksekusi saat run program.

Python mengenal dua jenis komentar, yaitu komentar satu baris dan multi-baris.

A.3.1. Komentar satu baris

Karakter `#` digunakan untuk menuliskan komentar, contoh:

```
# ini adalah komentar  
print("halo,")  
print("selamat pagi!") # ini juga komentar  
  
# println("statement ini tidak akan dipanggil")
```

Jika di-run, outputnya:

```
✓ TERMINAL  
  
PS D:\Labs\komentar> python .\main.py  
halo,  
selamat pagi!
```

Bisa dilihat statement yang diawali dengan tanda `#` tidak dieksekusi.

A.3.2. Komentar multi-baris

Komentar multi-baris bisa diterapkan melalui dua cara:

● Komentar menggunakan

```
# ini adalah komentar  
# ini juga komentar  
# komentar baris ke-3
```

● Komentar menggunakan `"""` atau `' '`

Karakter `"""` atau `' '` sebenarnya digunakan untuk membuat *multiline string* atau string banyak baris. Selain itu, bisa juga dipergunakan sebagai penanda komentar multi baris. Contoh penerapannya:

```
"""  
ini adalah komentar  
ini juga komentar  
komentar baris ke-3  
"""
```

Atau bisa juga ditulis seperti ini untuk komentar satu baris:

```
"""ini adalah komentar"""
```

- Lebih detailnya mengenai string dibahas pada chapter *String dan*

Catatan chapter

● Source code praktik

github.com/novalagung/dasarpemrogramanpython-example/..../komentar

● Referensi

- <https://docs.python-guide.org/writing/documentation/>

A.4. Variabel

Dalam konsep programming, variabel adalah suatu nama yang dikenali komputer sebagai penampung nilai/data yang disimpan di memory. Sebagai contoh nilai `3.14` disimpan di variabel bernama `PI`.

Pada chapter ini kita akan belajar tentang penerapan variabel di Python.

A.4.1. Deklarasi variabel

Agar dikenali oleh komputer, variabel harus dideklarasikan. Deklarasi variabel di Python cukup sederhana, caranya tinggal tulis saja nama variabel kemudian diikuti operator *assignment* beserta nilai awal yang ingin dimasukkan ke variabel tersebut. Contoh:

```
nama = "noval"  
hobi = 'makan'  
umur = 18  
laki = True
```

Nilai string (`str`) bisa dituliskan dengan menggunakan literal `"` ataupun `'`

Selanjutnya, coba kita munculkan nilai ke-empat variabel di atas ke layar menggunakan statement `print`. Caranya:

```
print("==== biodata ====")
print("nama: %s" % (nama))
print("hobi: %s, umur: %d, laki: %r" % (hobi, umur, laki))
```

```
▼ TERMINAL

PS D:\Labs\variables> python main.py
==== biodata ====
nama: noval
hobi: makan, umur: 18, laki: True
```

● **Output formatting** `print`

Statement berikut adalah contoh cara memunculkan string ke layar output (`stdout`):

```
print("==== biodata ====")
```

Sedangkan contoh berikut adalah penerapan teknik *output formatting* untuk mem-format string ke layar output:

```
print("nama: %s" % (nama))
# output => "nama: noval"
```

Karakter `%s` disitu akan di-replace dengan nilai variabel `nama` sebelum dimunculkan. Dan `%s` disini menandakan bahwa data yang akan me-replace-nya bertipe data `string`.

Selain `%s` ada juga `%d` untuk data bertipe numerik integer, dan `%r` untuk data bertipe `bool`.

```
print("hobi: %s, umur: %d, laki: %r" % (hobi, umur, laki))  
# output => "hobi: makan, umur: 18, laki: True"
```

Lebih detailnya mengenai output formatting dibahas terpisah pada chapter *Output formatting*

A.4.2. Naming convention variabel

Mengacu ke dokumentasi [PEP 8 – Style Guide for Python Code](#), nama variabel dianjurkan untuk menggunakan `snake_case`.

```
pesan = 'halo, selamat pagi'  
nilai_ujian = 99.2
```

A.4.3. Operasi assignment

Penulisan statement operasi *assignment* sama seperti statement deklarasi variabel.

```
nama = "noval"  
umur = 18  
nama = "noval agung"  
umur = 21
```

A.4.4. Deklarasi variabel beserta tipe data

Tipe data variabel bisa ditentukan secara eksplisit, penulisannya bisa dilihat pada kode berikut:

```
nama: str = "noval"  
hobi: str = 'makan'  
umur: int = 18  
laki: bool = True  
nilai_ujian: float = 99.2
```

*Lebih detailnya mengenai tipe data dibahas terpisah pada chapter **Tipe Data***

A.4.5. Deklarasi banyak variabel sebaris

Contoh penulisan deklarasi banyak variabel dalam satu baris bisa dilihat pada kode berikut:

```
nilai1, nilai2, nilai3, nilai4 = 24, 25, 26, 21  
nilai_rata_rata = (nilai1 + nilai2 + nilai3 + nilai4) / 4  
  
print("rata-rata nilai: %f" % (nilai_rata_rata))
```

Karakter `%f` digunakan untuk mem-format nilai `float`

Output program di atas:

```
✓ TERMINAL  
  
PS D:\Labs\variables> python main.py  
rata-rata nilai: 24.000
```

Catatan chapter

● Source code praktik

github.com/novalagung/dasarpemrogramanpython-example/./variables

● Referensi

- https://www.w3schools.com/python/python_datatypes.asp
- <https://peps.python.org/pep-0008/>
- https://en.wikipedia.org/wiki/Snake_case
- https://www.learnpython.org/en/String_Formatting

A.5. Konstanta

Konstanta adalah sebuah variabel yang nilainya didefinisikan di awal dan tidak bisa diubah. Pada chapter ini kita akan mempelajari tentang penerapan Konstanta di Python.

A.5.1. Konstanta di Python

Deklarasi konstanta (atau sebuah nilai konstan yang tidak bisa diubah setelah didefinisikan) di Python bisa dilakukan menggunakan bantuan tipe *class* bernama `typing.Final`.

Untuk menggunakannya, `typing.Final` perlu di-import terlebih dahulu menggunakan statement `from` dan `import`.

```
from typing import Final
```

```
PI: Final = 3.14  
print("pi: %f" % (PI))
```

▼ TERMINAL

```
PS D:\Labs\variables> python main.py  
pi: 3.140000
```

🕒 Module import

Keyword `import` digunakan untuk meng-import sesuatu, sedangkan keyword

`from` digunakan untuk menentukan dari module mana sesuatu tersebut akan di-import.

Lebih detailnya mengenai `import` dan `from` dibahas terpisah pada chapter [Module Import](#)

Statement `from typing import Final` artinya adalah meng-import tipe `Final` dari module `typing` yang dimana module ini merupakan bagian dari Python standard library (stdlib).

Lebih detailnya mengenai Python standard library (stdlib) dibahas terpisah pada chapter [Python standard library \(stdlib\)](#)

A.5.2. Tipe *class* `typing.Final`

Tipe `Final` digunakan untuk menandai suatu variabel adalah tidak bisa diubah nilainya (konstanta). Cara penerapan `Final` bisa dengan dituliskan tipe data konstanta-nya secara eksplisit, atau tidak ditentukan.

```
# tipe konstanta PI tidak ditentukan secara eksplisit,  
# melainkan didapat dari tipe data nilai  
PI: Final = 3.14  
  
# tipe konstanta TOTAL_MONTH ditentukan secara eksplisit yaitu  
`int`  
TOTAL_MONTH: Final[int] = 12
```

Lebih detailnya mengenai tipe data dibahas terpisah pada chapter [Tipe](#)

A.5.3. *Naming convention* konstanta

Mengacu ke dokumentasi [PEP 8 – Style Guide for Python Code](#), nama konstanta harus dituliskan dalam huruf besar (UPPER_CASE).

Catatan chapter

● Source code praktik

```
github.com/novalagung/dasarpemrogramanpython-example/../../konstanta
```

● Referensi

- <https://docs.python.org/3/library/typing.html#typing.Final>
- <https://peps.python.org/pep-0008/>

A.6. Tipe Data

Python mengenal cukup banyak tipe data, mulai dari yang *built-in* (atau bawaan) maupun custom type. Pada chapter ini kita akan mempelajari *high-level overview* mengenai tipe-tipe tersebut.

A.6.1. Tipe data numerik

Ada setidaknya 3 tipe data numerik di Python, yaitu:

Tipe data	Keterangan	Contoh
<code>int</code>	menampung bilangan bulat atau <i>integer</i>	<code>number_1 = 10000024</code>
<code>float</code>	menampung bilangan desimal atau <i>floating point</i>	<code>number_2 = 3.14</code>
<code>complex</code>	menampung nilai berisi kombinasi bilangan real dan imajiner	<code>number_3 = 120+3j</code>

A.6.2. Tipe data `str`

Tipe string direpresentasikan oleh `str`, pembuatannya bisa menggunakan

literal string yang ditandai dengan tanda awalan dan akhiran tanda `"` atau `'`.

- Menggunakan tanda petik dua (`"`)

```
# string sebaris
string_1 = "hello python"

# string multi-baris
string_2 = """Selamat
Belajar
Python"""
```

- Menggunakan tanda petik satu (`'`)

```
# string sebaris
string_3 = 'for the horde!'

# string multi-baris
string_4 = '''
Sesuk
Preiiii
'''
```

Jika ada baris baru (atau *newline*) di bagian awal penulisan `'''` atau `"""` maka baris baru tersebut merupakan bagian dari string. Jika ingin meng-exclude-nya bisa menggunakan `"""\"` atau `'''\"`. Contoh:

```
string_5 = '''\
Sesuk
Preiiii
'''
```

Lebih detailnya mengenai string dibahas pada chapter *String dan Operasi String*

A.6.3. Tipe data `bool`

Literal untuk tipe data boolean di Python adalah `True` untuk nilai benar, dan `False` untuk nilai salah.

```
bool_1 = True
bool_2 = False
```

A.6.4. Tipe data list

List adalah tipe data di Python untuk menampung nilai kolektif yang disimpan secara urut, dan isinya bisa banyak varian tipe data (tidak harus sejenis). Cara penerapan list adalah dengan menuliskan nilai kolektif dengan pembatas `,` dan diapit tanda `[` dan `]`.

```
# list with int as element's data type
list_1 = [2, 4, 8, 16]

# list with str as element's data type
list_2 = ["grayson", "jason", "tim", "damian"]

# list with various data type in the element
list_3 = [24, False, "Hello Python"]
```

Pengaksesan element list menggunakan notasi `list[index_number]`. Contoh:

```
list_1 = [2, 4, 8, 16]
print(list_1[2])
# output: 8
```

Lebih detailnya mengenai list dibahas pada chapter *Tipe Data List*

A.6.5. Tipe data tuple

Tuple adalah tipe data kolektif yang mirip dengan list, dengan perbedaan adalah:

- Nilai pada data list adalah bisa diubah (*mutable*), sedangkan nilai data `tuple` tidak bisa diubah (*immutable*).
- List menggunakan tanda `[` dan `]` untuk penulisan literal, sedangkan pada tuple yang digunakan adalah tanda `(` dan `)`.

```
# tuple with int as element's data type
tuple_1 = (2, 3, 4)

# tuple with str as element's data type
tuple_2 = ("numenor", "valinor")

# tuple with various data type in the element
tuple_3 = (24, False, "Hello Python")
```

Pengaksesan element tuple menggunakan notasi `tuple[index_number]`.

Contoh:

```
tuple_1 = (2, 3, 4)
```

Lebih detailnya mengenai tuple dibahas pada chapter *Tipe Data Tuple*

A.6.6. Tipe data dictionary

Tipe data `dict` atau dictionary berguna untuk menyimpan data kolektif terstruktur berbentuk *key value*. Contoh penerapan:

```
profile_1 = {  
    "name": "Noval",  
    "is_male": False,  
    "age": 16,  
    "hobbies": ["gaming", "learning"]  
}
```

Pengaksesan property dictionary menggunakan notasi `dict[property_name]`.
Contoh:

```
print("name: %s" % (profile_1["name"]))  
print("hobbies: %s" % (profile_1["hobbies"]))
```

Penulisan data dictionary diperbolehkan secara horizontal, contohnya seperti berikut:

```
profile_1 = { "name": "Noval", "hobbies": ["gaming", "learning"] }
```

Lebih detailnya mengenai dictionary dibahas pada chapter *Tipe Data Dictionary & Sets*

A.6.7. Tipe data sets

Tipe data sets adalah cara lain untuk menyimpan data kolektif. Tipe data ini memiliki beberapa kelemahan:

- Tidak bisa menyimpan informasi urutan data
- Elemen data yang sudah didefinisikan tidak bisa diubah nilainya (tapi bisa dihapus)
- Tidak bisa diakses menggunakan index (tetapi bisa menggunakan perulangan)

Contoh penerapan sets:

```
set_1 = {"pineapple", "spaghetti"}  
print(set_1)
```

Lebih detailnya mengenai sets dibahas pada chapter [Tipe Data Dictionary & Sets](#)

A.6.8. Tipe data lainnya

Selain tipe-tipe di atas ada juga beberapa tipe data lainnya, seperti frozenset, bytes, memoryview, range; dan kesemuanya akan dibahas satu per satu di chapter terpisah.

Catatan chapter

● Source code praktik

[github.com/novalagung/dasarpemrogramanpython-example/./tipe-data](https://github.com/novalagung/dasarpemrogramanpython-example/tree/master/tipe-data)

● Referensi

- <https://docs.python.org/3/tutorial/introduction.html>
- <https://www.digitalocean.com/community/tutorials/python-data-types>
- <https://note.nkmk.me/en/python-int-max-value/>

A.7. Operator

Operator adalah suatu karakter yang memiliki kegunaan khusus contohnya seperti `+` untuk operasi aritmatika tambah, dan `and` untuk operasi logika **AND**. Pada chapter ini kita akan mempelajari macam-macam operator yang ada di Python.

A.7.1. Operator aritmatika

Operator	Keterangan	Contoh
<code>+</code>	operasi tambah	<code>num = 2 + 2</code> → hasilnya <code>num</code> nilainya <code>4</code>
unary <code>+</code>	penanda nilai positif	<code>num = +2</code> → hasilnya <code>num</code> nilainya <code>2</code>
<code>-</code>	operasi pengurangan	<code>num = 3 - 2</code> → hasilnya <code>num</code> nilainya <code>1</code>
unary <code>-</code>	penanda nilai negatif	<code>num = -2</code> → hasilnya <code>num</code> nilainya <code>-2</code>
<code>*</code>	operasi perkalian	<code>num = 3 * 3</code> → hasilnya <code>num</code> nilainya <code>9</code>

Operator	Keterangan	Contoh
/	operasi pembagian	<code>num = 8 / 2</code> → hasilnya <code>num</code> nilainya 4
//	operasi bagi dengan hasil dibulatkan ke bawah	<code>num = 10 // 3</code> → hasilnya <code>num</code> nilainya 3
%	operasi modulo (pencarian sisa hasil bagi)	<code>num = 7 % 4</code> → hasilnya <code>num</code> nilainya 3
**	operasi pangkat	<code>num = 3 ** 2</code> → hasilnya <code>num</code> nilainya 9

A.7.2. Operator *assignment*

Operator assignment adalah `=`, digunakan untuk operasi assignment (atau penentuan nilai) sekaligus untuk deklarasi variabel jika variabel tersebut sebelumnya belum terdeklarasi. Contoh:

```
num_1 = 12
num_2 = 24

num_2 = 12
num_3 = num_1 + num_2

print(num_3)
# output: 24
```

A.7.3. Operator perbandingan

Operator perbandingan pasti menghasilkan nilai kebenaran `bool` dengan kemungkinan hanya dua nilai, yaitu benar (`True`) atau salah (`False`).

Python mengenal operasi perbandingan standar yang umumnya juga dipakai di bahasa lain.

Operator	Keterangan	Contoh
<code>==</code>	apakah kiri sama dengan kanan	<code>res = 4 == 5</code> → hasilnya <code>res</code> nilainya <code>False</code>
<code>!=</code>	apakah kiri tidak sama dengan kanan	<code>res = 4 != 5</code> → hasilnya <code>res</code> nilainya <code>True</code>
<code>></code>	apakah kiri lebih besar dibanding kanan	<code>res = 4 > 5</code> → hasilnya <code>res</code> nilainya <code>False</code>
<code><</code>	apakah kiri lebih kecil dibanding kanan	<code>res = 4 < 5</code> → hasilnya <code>res</code> nilainya <code>True</code>
<code>>=</code>	apakah kiri lebih besar atau sama dengan kanan	<code>res = 5 >= 5</code> → hasilnya <code>res</code> nilainya <code>True</code>
<code><=</code>	apakah kiri lebih kecil atau sama dengan kanan	<code>res = 4 <= 5</code> → hasilnya <code>res</code> nilainya <code>False</code>

A.7.4. Operator logika

Operator	Keterangan	Contoh
<code>and</code>	operasi logika AND	<code>res = (4 == 5) and (2 != 3) →</code> hasilnya <code>res</code> nilainya <code>False</code>
<code>or</code>	operasi logika OR	<code>res = (4 == 5) or (2 != 3) →</code> hasilnya <code>res</code> nilainya <code>True</code>
<code>not</code> atau <code>!</code>	operasi logika negasi (atau NOT)	<code>res = not (2 == 3) →</code> hasilnya <code>res</code> nilainya <code>True</code> <code>res = !(2 == 3) →</code> hasilnya <code>res</code> nilainya <code>True</code>

A.7.5. Operator bitwise

Operator	Keterangan	Contoh
<code>&</code>	operasi bitwise AND	<code>x & y = 0 (0000 0000)</code>
<code> </code>	operasi bitwise OR	<code>x y = 14 (0000 1110)</code>
<code>~</code>	operasi bitwise NOT	<code>~x = -11 (1111 0101)</code>
<code>^</code>	operasi bitwise XOR	<code>x ^ y = 14 (0000 1110)</code>

Operator	Keterangan	Contoh
>>	operasi bitwise right shift	x >> 2 = 2 (0000 0010)
<<	operasi bitwise left shift	x << 2 = 40 (0010 1000)

A.7.6. Operator *identity* (`is`)

Operator `is` memiliki kemiripan dengan operator logika `==`, perbedaannya pada operator `is` yang dibandingkan bukan nilai, melainkan identitas atau ID-nya.

Bisa saja ada 2 variabel bernilai sama tapi identitasnya berbeda. Contoh:

```
num_1 = 100001
num_2 = 100001

res = num_1 is num_2
print("num_1 is num_2 =", res)
print("id(num_1): %s, id(num_2): %s" % (id(num_1), id(num_2)))
```

▼ TERMINAL

```
PS D:\Labs\operator> python.exe main.py
num_1 is num_2 = True
id(num_1): 2545659797168, id(num_2): 2545659797168
```

DANGER

Di Python ada *special case* yang perlu kita ketahui perihal penerapan operator `is` untuk operasi perbandingan identitas khusus tipe data

numerik. Silakan cek <https://stackoverflow.com/a/15172182/1467988> untuk lebih jelasnya.

● Fungsi `print()` tanpa output formatting

Statement `print("num_1 is not num_2 =", res)` adalah salah satu cara untuk printing data tanpa menggunakan output formatting (seperti `%s`).

Yang terjadi pada statement tersebut adalah, semua nilai argument pemanggilan fungsi `print()` akan digabung dengan delimiter `" "` kemudian ditampilkan ke layar console.

Agar lebih jelas, silakan perhatikan statement berikut, keduanya adalah menghasilkan output yang sama.

```
print("message: %s %s %s" % ("hello", "python", "learner"))  
print("message:", "hello", "python", "learner")
```

▼ TERMINAL

```
PS D:\Labs\operator> python.exe main.py  
message: hello python learner  
message: hello python learner
```

● Fungsi `id()`

Digunakan untuk mengambil nilai identitas atau ID suatu data. Contoh penerapannya sangat mudah, cukup panggil fungsi dan tulis data yang ingin diambil ID-nya sebagai argument pemanggilan fungsi.

```
data_1 = "hello world"
id_data_1 = id(data_1)

print("data_1:", data_1)          # hello world
print("id_data_1:", id_data_1)    # 19441xxxxxxxx
```

Nilai kembalian fungsi `id()` bertipe numerik.

A.7.7. Operator *membership* (`in`)

Operator `in` digunakan untuk mengecek apakah suatu nilai merupakan bagian dari data kolektif atau tidak.

Operator ini bisa dipergunakan pada semua tipe data kolektif seperti dictionary, sets, tuple, dan list. Selain itu, operator `in` juga bisa digunakan pada `str` untuk pengecekan substring

```
sample_list = [2, 3, 4]
is_3_exists = 3 in sample_list
print(is_3_exists)
# False

sample_tuple = ("hello", "python")
is_hello_exists = "hello" in sample_tuple
print(is_hello_exists)
# True

sample_dict = { "nama": "noval", "age": 12 }
is_key_nama_exists = "nama" in sample_dict
print(is_key_nama_exists)
# True
```


Operator `in` jika diterapkan pada tipe dictionary, yang di-check adalah key-nya bukan value-nya.

Catatan chapter

● Source code praktik

[github.com/novalagung/dasarpemrogramanpython-example/..../operator](https://github.com/novalagung/dasarpemrogramanpython-example/blob/master/operator)

● Chapter relevan lainnya

- Variabel
- Tipe Data

● Referensi

- <https://realpython.com/python-operators-expressions/>
- <https://www.programiz.com/python-programming/operators>
- <https://stackoverflow.com/a/15172182/1467988>

A.8. Seleksi kondisi → if, elif, else

Seleksi kondisi adalah suatu blok kode yang dieksekusi hanya ketika kriteria yang ditentukan terpenuhi. Seleksi kondisi banyak digunakan untuk kontrol alur program.

Python mengenal beberapa keyword seleksi kondisi, yang pada chapter ini akan kita pelajari.

A.8.1. Keyword `if`

`if` adalah keyword seleksi kondisi di Python. Cara penerapannya sangat mudah, cukup tulis saja keyword tersebut, kemudian diikuti dengan kondisi berupa nilai `bool` atau statement operasi logika, lalu dibawahnya ditulis blok kode yang ingin dieksekusi ketika kondisi tersebut terpenuhi. Contoh:

```
grade = 100

if grade == 100:
    print("perfect")

if grade == 90:
    print("ok")
    print("keep working hard!")
```

```
✓ TERMINAL

PS D:\Labs\if-elif-else> python.exe main.py
perfect
```

Bisa dilihat di output, hanya pesan `perfect` yang muncul karena kondisi `grade == 100` adalah yang terpenuhi. Sedangkan statement `print("ok")` tidak tereksekusi karena nilai variabel `grade` bukanlah `90`.

● **Block indentation**

Di python, suatu blok kondisi ditandai dengan *indentation* atau spasi, yang menjadikan kode semakin menjorok ke kanan.

Sebagai contoh, 2 blok kode `print` berikut merupakan isi dari seleksi kondisi `if grade == 90`.

```
if grade == 90:
    print("ok")
    print("keep working hard!")
```

Sesuai aturan *PEP 8 – Style Guide for Python Code*, indentation di Python menggunakan 4 karakter spasi dan bukan karakter tab.

A.8.2. Keyword `elif`

`elif` (kependekan dari **else if**) digunakan untuk menambahkan blok seleksi kondisi baru, untuk mengantisipasi blok `if` yang tidak terpenuhi.

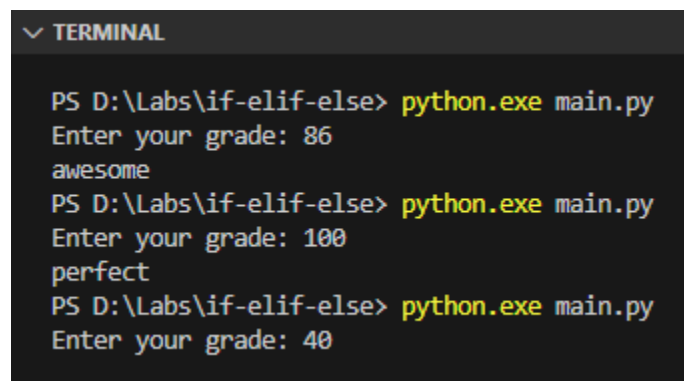
Dalam penerapannya, suatu blok seleksi kondisi harus diawali dengan `if`.

Keyword `elif` hanya bisa dipergunakan pada kondisi setelahnya yang masih satu rantai (atau *chain*). Contoh:

```
str_input = input('Enter your grade: ')
grade = int(str_input)

if grade == 100:
    print("perfect")
elif grade >= 85:
    print("awesome")
elif grade >= 65:
    print("passed the exam")
```

Jalankan program di atas, kemudian inputkan suatu nilai numerik lalu tekan enter.



```

✓ TERMINAL

PS D:\Labs\if-elif-else> python.exe main.py
Enter your grade: 86
awesome
PS D:\Labs\if-elif-else> python.exe main.py
Enter your grade: 100
perfect
PS D:\Labs\if-elif-else> python.exe main.py
Enter your grade: 40
```

Kode di atas menghasilkan:

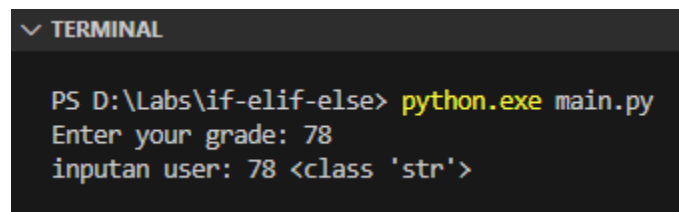
- Ketika nilai inputan adalah `86`, muncul pesan `awesome` karena blok seleksi kondisi yang terpenuhi adalah `elif grade >= 85`.
- Ketika nilai inputan adalah `100`, muncul pesan `perfect` karena blok seleksi kondisi yang terpenuhi adalah `grade == 100`.
- Ketika nilai inputan adalah `40`, tidak muncul pesan karena semua blok seleksi kondisi tidak terpenuhi.

● Fungsi `input()`

Fungsi `input` digunakan untuk menampilkan suatu pesan text (yang disisipkan saat fungsi dipanggil) dan mengembalikan nilai inputan user dalam bentuk string.

Agar makin jelas, silakan praktekan kode berikut:

```
str_input = input('Enter your grade: ')\nprint("inputan user:", str_input, type(str_input))
```



```
▼ TERMINAL\n\nPS D:\Labs\if-elif-else> python.exe main.py\nEnter your grade: 78\ninputan user: 78 <class 'str'>
```

Kode di atas menghasilkan:

1. Text `Enter your grade :` muncul, kemudian kursor akan berhenti disitu.
2. User perlu menuliskan sesuatu kemudian menekan tombol enter agar eksekusi program berlanjut.
3. Inputan dari user kemudian menjadi nilai balik fungsi `input()` (yang pada contoh di atas ditangkap oleh variabel `input_str`).
4. Nilai inputan user di print menggunakan statement `print("inputan user:", str_input)`.

● Fungsi `type()`

Fungsi `type()` digunakan untuk melihat informasi tipe data dari suatu nilai atau variabel. Fungsi ini mengembalikan string dalam format `<class`

```
'tipe_data'>.
```

◎ **Type conversion / konversi tipe data**

Konversi tipe data `str` ke `int` dilakukan menggunakan fungsi `int()`. Dengan menggunakan fungsi tersebut, data string yang disisipkan pada parameter, tipe datanya berubah menjadi `int`.

Sebagai contoh, bisa dilihat pada program berikut ini, hasil statement `type(grade)` adalah `<class 'int'>` yang menunjukkan bahwa tipe datanya adalah `int`.

```
str_input = input('Enter your grade: ')
grade = int(str_input)
print("inputan user:", grade, type(grade))
```

▼ TERMINAL

```
PS D:\Labs\if-elif-else> python.exe main.py
Enter your grade: 79
inputan user: 79 <class 'int'>
```

*Lebih detailnya mengenai mengenai type conversion dibahas pada chapter **Konversi Tipe Data***

A.8.3. Keyword `else`

`else` digunakan sebagai blok seleksi kondisi penutup ketika blok `if` dan/atau `elif` dalam satu *chain* tidak ada yang terpenuhi. Contoh:

```
str_input = input('Enter your grade: ')
grade = int(str_input)

if grade == 100:
    print("perfect")
elif grade >= 85:
    print("awesome")
elif grade >= 65:
    print("passed the exam")
else:
    print("below the passing grade")
```

A.8.4. Seleksi kondisi bercabang / *nested*

Seleksi kondisi bisa saja berada di dalam suatu blok seleksi kondisi. Teknik ini biasa disebut dengan seleksi kondisi bercabang atau bersarang.

Di Python, cara penerapannya cukup dengan menuliskan blok seleksi kondisi tersebut. Gunakan *indentation* yang lebih ke kanan untuk seleksi kondisi terdalam.

```
str_input = input('Enter your grade: ')
grade = int(str_input)

if grade == 100:
    print("perfect")

elif grade >= 85:
    print("awesome")

elif grade >= 65:
```

▼ TERMINAL

```
PS D:\Labs\if-elif-else> python.exe main.py
Enter your grade: 69
passed the exam
but you need to improve it!
```

Pada kode di atas, pada seleksi kondisi terluar, di bawah blok `if` dan `elif` sengaja penulis tulis di baris baru agar lebih mudah untuk dibaca. Hal seperti ini diperbolehkan.

A.8.5. Seleksi kondisi dengan operasi logika

Keyword `and`, `or`, dan `not` bisa digunakan dalam seleksi kondisi.

Contohnya:

```
grade = int(input('Enter your current grade: '))
prev_grade = int(input('Enter your previous grade: '))

if grade >= 90 and prev_grade >= 65:
    print("awesome")
if grade >= 90 and prev_grade < 65:
    print("awesome. you definitely working hard, right?")
elif grade >= 65:
    print("passed the exam")
else:
    print("below the passing grade")

if (grade >= 65 and not prev_grade >= 65) or (not grade >= 65 and
prev_grade >= 65):
    print("at least you passed one exam. good job!")
```


A.8.6. Seleksi kondisi sebaris & *ternary*

Silakan perhatikan kode berikut:

```
if grade >= 65:  
    print("passed the exam")  
else:  
    print("below the passing grade")
```

Kode di atas bisa dituliskan menggunakan beberapa metode:

● ***One-line / sebaris***

```
if grade >= 65: print("passed the exam")  
if grade < 65: print("below the passing grade")
```

Metode penulisan ini bisa diterapkan pada blok kode seleksi kondisi yang hanya memiliki 1 kondisi.

● ***Ternary***

```
print("passed the exam") if grade >= 65 else print("below the  
passing grade")
```

Metode penulisan ini bisa diterapkan pada blok kode seleksi kondisi yang memiliki 2 kondisi (`True` dan `False`).

● Ternary dengan nilai balik

```
message = "passed the exam" if grade >= 65 else "below the  
passing grade"  
print(message)
```

Metode penulisan ini sebetulnya sama seperti yang di atas, perbedaannya adalah setiap kondisi menghasilkan nilai balik (yang kemudian ditangkap variabel `message`).

Catatan chapter

● Source code praktik

```
github.com/novalagung/dasarpemrogramanpython-example/./if-elif-  
else
```

● Referensi

- <https://docs.python.org/3/tutorial/controlflow.html>



A.9. Perulangan → for, range

Perulangan atau *loop* merupakan teknik untuk mengulang-ulang eksekusi suatu blok kode. Pada chapter ini kita akan mempelajari penerapannya di Python.

A.9.1. Keyword `for` dan fungsi `range()`

Perulangan di Python bisa dibuat menggunakan kombinasi keyword `for` dan fungsi `range()`.

- keyword `for` adalah keyword untuk perulangan, dalam penerapannya diikuti dengan keyword `in`.
- fungsi `range()` digunakan untuk membuat object *range*, yang umumnya dipakai sebagai kontrol perulangan.

Agar lebih jelas, silakan perhatikan dan test kode berikut:

```
for i in range(5):  
    print("index:", i)
```

```
for i in range(5):
    print("index:", i)
✓ 0.0s
... index: 0
      index: 1
      index: 2
      index: 3
      index: 4
```

Statement `print("index:", i)` muncul 5 kali, karena perulangan dilakukan dengan kontrol `range(5)` dimana fungsi ini menciptakan object *range* dengan isi deret angka sejumlah 5 dimulai dari angka 0 hingga 4.

Statement `for i in range(5):` adalah contoh penulisan perulangan menggunakan `for` dan `range()`. Variabel `i` berisi nilai *counter* setiap iterasi, yang pada konteks ini adalah angka 0 hingga 4.

Statement `print("index:", i)` wajib ditulis menjorok ke kanan karena merupakan isi dari blok perulangan `for i in range(5):`.

● Fungsi `list()`

Fungsi `range()` menghasilkan object *sequence*, yaitu suatu data yang strukturnya mirip seperti list. Object tersebut bisa ditampilkan dalam bentuk list dengan cara membungkusnya menggunakan fungsi `list()` (konversi tipe data *range* ke list).

```
r = range(5)
print("r:", list(r))
```

```
r = range(5)
print("r:", list(r))
✓ 0.0s
... r: [0, 1, 2, 3, 4]
```

- Lebih detailnya mengenai list dibahas pada chapter [List](#)
- Lebih detailnya mengenai mengenai type conversion dibahas pada chapter [Konversi Tipe Data](#)

A.9.2. Penerapan fungsi `range()`

Statement `range(n)` menghasilkan data *range* sejumlah `n` yang isinya dimulai dari angka `0`. Syntax `range(n)` adalah bentuk paling sederhana penerapan fungsi ini.

Selain `range(n)` ada juga beberapa cara penulisan lainnya:

- Menggunakan `range(start, stop)`. Hasilnya data *range* dimulai dari `start` dan hingga `stop - 1`. Sebagai contoh, `range(1, 4)` menghasilkan data range `[1, 2, 3]`.
- Menggunakan `range(start, stop, step)`. Hasilnya data *range* dimulai dari `start` dan hingga `stop - 1`, dengan nilai *increment* sejumlah `step`. Sebagai contoh, `range(1, 10, 3)` menghasilkan data range `[1, 4, 7]`.

Agar lebih jelas, silakan perhatikan kode berikut. Ke-3 perulangan ini ekuivalen, menghasilkan output yang sama.

```
for i in range(3):  
    print("index:", i)
```

▼ TERMINAL

```
PS D:\Labs\for-range> python.exe main.py  
index: 0  
index: 1  
index: 2
```

```
for i in range(0, 3):  
    print("index:", i)
```

▼ TERMINAL

```
PS D:\Labs\for-range> python.exe main.py  
index: 0  
index: 1  
index: 2
```

Tambahan contoh penerapan `for` dan `range()` :

```
for i in range(2, 10, 2):  
    print("index:", i)
```

```
for i in range(2, 10, 2):  
    print("index:", i)  
✓ 0.0s  
... index: 2  
index: 4  
index: 6  
index: 8
```

```
for i in range(5, -5, -1):  
    print("index:", i)
```

```
for i in range(5, -5, -1):  
    print("index:", i)  
✓ 0.0s  
... index: 5  
index: 4  
index: 3  
index: 2  
index: 1  
index: 0  
index: -1  
index: -2  
index: -3  
index: -4
```

A.9.3. Perulangan `for` tanpa `range()`

Perulangan menggunakan `for` bisa dilakukan pada beberapa jenis tipe data (seperti list, string, tuple, dan lainnya) caranya dengan langsung menuliskan saja variabel atau data tersebut pada statement `for`. Contoh penerapannya bisa dilihat di bawah ini:

● Iterasi data list

```
messages = ["morning", "afternoon", "evening"]
for m in messages:
    print(m)
```

▼ TERMINAL

```
PS D:\Labs\for-range> python.exe main.py
morning
afternoon
evening
```

● Iterasi data tuple

```
numbers = ("twenty four", 24)
for n in numbers:
    print(n)
```

```
✓ TERMINAL
PS D:\Labs\for-range> python.exe main.py
twenty four
24
```

● Iterasi data string

Penggunaan keyword `for` pada tipe data `str` (atau string) akan mengiterasi setiap karakter yang ada di string.

```
for char in "hello python":
    print(char)
```

```
✓ TERMINAL
PS D:\Labs\for-range> python.exe main.py
h
e
l
l
o

p
y
t
h
o
n
```

● Iterasi data dictionary

Penggunaan keyword `for` pada tipe data `dict` (atau dictionary) akan mengiterasi *key*-nya. Dari *key* tersebut *value* bisa diambil dengan mudah menggunakan notasi `dict[key]`.


```
bio = {  
    "name": "toyota camry",  
    "year": 1993,  
}  
  
for key in bio:  
    print("key:", key, "value:", bio[key])
```

▼ TERMINAL

```
PS D:\Labs\for-range> python.exe main.py  
key: name value: toyota camry  
key: year value: 1993
```

● Iterasi data sets

```
numbers = {"twenty four", 24}  
for n in numbers:  
    print(n)
```

▼ TERMINAL

```
PS D:\Labs\for-range> python.exe main.py  
24  
twenty four
```

A.9.4. Perulangan bercabang / *nested* for

Cara penerapan *nested loop* adalah cukup dengan menuliskan statement `for` sebagai isi dari statement `for` atasnya. Contoh:

```
max = int(input("jumlah bintang: "))

for i in range(max):
    for j in range(0, max - i):
        print("*", end=" ")
    print()
```

```
✓ TERMINAL

PS D:\Labs\for-range> python.exe main.py
jumlah bintang: 3
* * *
* *
*

PS D:\Labs\for-range> python.exe main.py
jumlah bintang: 5
* * * * *
* * * *
* * *
* *
*
```

● Parameter opsional `end` pada fungsi `print()`

Fungsi `print()` memiliki parameter opsional bernama `end`, kegunaannya untuk mengubah karakter akhir yang muncul setelah data string di-*print*. *Default* nilai paramter `end` ini adalah `\n` atau karakter baris baru, itulah kenapa setiap selesai print pasti ada baris baru.

Statement `print("*", end=" ")` akan menghasilkan pesan `*` yang di-akhiri dengan karakter spasi karena nilai parameter `end` di-set dengan nilai karakter spasi (atau).

Lebih detailnya tentang fungsi dan parameter opsional dibahas pada

● Fungsi `print()` tanpa parameter

Pemanggilan fungsi `print()` argument/parameter menghasilkan baris baru.

Catatan chapter

● Source code praktik

```
github.com/novalagung/dasarpemrogramanpython-example/./for-range
```

● Chapter relevan lainnya

- List
- String
- Fungsi

● Referensi

- <https://docs.python.org/3/library/functions.html#func-range>
- <https://docs.python.org/3/library/functions.html#print>
- <https://python-reference.readthedocs.io/en/latest/docs/functions/range.html>

A.10. Perulangan → while

Di Python, selain keyword `for` ada juga keyword `while` yang fungsinya kurang lebih sama yaitu untuk perulangan. Bedanya, perulangan menggunakan `while` terkontrol via operasi logika atau nilai `bool`.

Pada chapter ini kita akan mempelajari cara penerapannya.

A.10.1. Keyword `while`

Cara penerapan perulangan ini adalah dengan menuliskan keyword `while` kemudian diikuti dengan nilai `bool` atau operasi logika. Contoh:

```
should_continue = True

while should_continue:
    n = int(input("enter an even number greater than 0: "))

    if n <= 0 or n % 2 == 1:
        print(n, "is not an even number greater than 0")
        should_continue = False
    else:
        print("number:", n)
```

▼ TERMINAL

```
PS D:\Labs> python.exe main.py
enter an even number greater than 0: 10
number: 10
enter an even number greater than 0: 2
number: 2
enter an even number greater than 0: 8
number: 8
enter an even number greater than 0: 7
7 is not an even number greater than 0
```

Program di atas memunculkan *prompt* inputan `enter an even number greater than 0:` yang dimana akan terus muncul selama user tidak menginputkan angka ganjil atau angka dibawah sama dengan `0`.

Contoh lain penerapan `while` dengan kontrol adalah operasi logika:

```
n = int(input("enter max data: "))
i = 0

while i < n:
    print("number", i)
    i += 1
```

▼ TERMINAL

```
PS D:\Labs> python.exe main.py
enter max data: 6
number 0
number 1
number 2
number 3
number 4
number 5
```

● Operasi *increment* dan *decrement*

Python tidak mengenal operator *unary* `++` dan `--`. Solusi untuk melakukan operasi *increment* maupun *decrement* bisa menggunakan cara berikut:

Operasi	Cara 1	Cara 2
<i>Increment</i>	<code>i += 1</code>	<code>i = i + 1</code>
<i>Decrement</i>	<code>i -= 1</code>	<code>i = i - 1</code>

A.10.2. Perulangan `while` vs `for`

Operasi `while` cocok digunakan untuk perulangan yang dimana kontrolnya adalah operasi logika atau nilai boolean yang tidak ada kaitannya dengan *sequence*.

Pada program yang sudah di tulis di atas, perulangan akan menjadi lebih ringkas dengan pengaplikasian keyword `for`, silakan lihat perbandingannya di bawah ini:

- Dengan keyword `while`:

```
n = int(input("enter max data: "))
i = 0

while i < n:
    print("number", i)
    i += 1
```

- Dengan keyword `for`:

```
n = int(input("enter max data: "))

for i in range(n):
    print("number", i)
```

Sedangkan keyword `for` lebih pas digunakan pada perulangan yang kontrolnya adalah data *range* atau data *sequence* seperti list atau lainnya.

A.10.3. Perulangan bercabang / *nested* `while`

Contoh perulangan bercabang bisa dilihat pada kode program berikut ini. Caranya cukup tulis saja keyword `while` di dalam block kode `while`.

```
n = int(input("enter max data: "))
i = 0

while i < n:
    j = 0

    while j < n - i:
        print("*", end=" ")
        j += 1

    print()
    i += 1
```

```
✓ TERMINAL
PS D:\Labs> python.exe main.py
enter max data: 4
* * * *
* * *
* *
*
PS D:\Labs> python.exe main.py
enter max data: 2
* *
*
```

A.10.4. Kombinasi `while` dan `for`

Kedua keyword perulangan yang sudah dipelajari, yaitu `for` dan `while` bisa dikombinasikan untuk membuat suatu *nested loop* atau perulangan bercabang.

Pada contoh berikut, kode program di atas diubah menggunakan kombinasi keyword `for` dan `while`.

```
n = int(input("enter max data: "))
i = 0

for i in range(n):
    j = 0

    while j < n - i:
        print("*", end=" ")
        j += 1

    print()
```


Catatan chapter

● Source code praktik

[github.com/novalagung/dasarpemrogramanpython-example/./while](https://github.com/novalagung/dasarpemrogramanpython-example/blob/master/while)

● Referensi

- <https://docs.python.org/3/tutorial/controlflow.html>

A.11. Perulangan → break, continue

Keyword `break` dan `continue` sering dipergunakan dalam perulangan untuk alterasi flow secara paksa, seperti memberhentikan perulangan atau memaksa perulangan untuk lanjut ke iterasi berikutnya.

Pada chapter ini kita akan mempelajarinya.

A.11.1. Keyword `break`

Pengaplikasian `break` biasanya dikombinasikan dengan seleksi kondisi. Sebagai contoh program sederhana berikut, yaitu program dengan spesifikasi:

- Berisi perulangan yang sifatnya berjalan terus-menerus tanpa henti (karenam menggunakan nilai `True` sebagai kontrol).
- Perulangan hanya berhenti jika nilai `n` (yang didapat dari inputan user) adalah tidak bisa dibagi dengan angka `3`.

```
while True:
    n = int(input("enter a number divisible by 3: "))
    if n % 3 != 0:
        break

    print("%d is divisible by 3" % (n))
```

▼ TERMINAL

```
PS D:\Labs> python.exe main.py
enter a number divisible by 3: 9
9 is divisible by 3
enter a number divisible by 3: 24
24 is divisible by 3
enter a number divisible by 3: 11
```

A.11.2. Keyword `continue`

Keyword `continue` digunakan untuk memaksa perulangan lanjut ke iterasi berikutnya (seperti proses skip).

Contoh penerapannya bisa dilihat pada kode berikut.

- Program berisi perulangan dengan kontrol adalah data *range* sebanyak 10 (dimana isinya adalah angka numerik dari `0` hingga `9`).
- Ketika nilai variabel counter `i` adalah dibawah `3` atau di atas `7` maka iterasi di-skip.

```
for i in range(10):
    if i < 3 or i > 7:
        continue
    print(i)
```

Efek dari `continue` adalah semua statement setelahnya akan di-skip. Pada program di atas, statement `print(i)` tidak dieksekusi ada `continue`.

Hasilnya bisa dilihat pada gambar berikut, nilai yang di-print adalah angka `3` hingga `7` saja.

```
✓ TERMINAL
PS D:\Labs> python.exe main.py
3
4
5
6
7
```

A.11.3. Label perulangan

Python tidak mengenal konsep perulangan yang memiliki label.

Teknik menamai perulangan dengan label umumnya digunakan untuk mengontrol flow pada perulangan bercabang / *nested*, misalnya untuk menghentikan perulangan terluar secara paksa ketika suatu kondisi terpenuhi.

Di Python, algoritma seperti ini bisa diterapkan namun menggunakan tambahan kode. Contoh penerapannya bisa dilihat pada kode berikut:

```
max = int(input("jumlah bintang: "))

outerLoop = True
for i in range(max):
    if not outerLoop:
        break

    for j in range(i + 1):
        print("*", end=" ")
        if j >= 7:
            outerLoop = False
            break
    print()
```

- Program yang memiliki perulangan *nested* dengan jumlah perulangan ada 2.

- Disiapkan sebuah variabel `bool` bernama `outerLoop` untuk kontrol perulangan terluar.
 - Ketika nilai `j` (yang merupakan variabel counter perulangan terdalam) adalah lebih dari atau sama dengan `7`, maka variabel `outerLoop` di set nilainya menjadi `False`, dan perulangan terdalam di-`break` secara paksa.
 - Dengan ini maka perulangan terluar akan terhenti.
-

Catatan chapter

● Source code praktik

```
github.com/novalagung/dasarpemrogramanpython-example/./break-continue
```

● Chapter relevan lainnya

- Perulangan → For
- Perulangan → While

● Referensi

- <https://docs.python.org/3/tutorial/controlflow.html>

A.12. List

List adalah tipe data kolektif yang disimpan secara urut dan bisa diubah nilainya.

Pada bahasa pemrograman umumnya ada tipe data **array**. List di Python ini memiliki banyak kemiripan dengan array, bedanya list bisa berisi data dengan berbagai macam tipe data, jadi tidak harus sejenis tipe datanya.

Pada chapter ini kita akan belajar lebih detail mengenai list dan pengoperasiannya.

A.12.1. Penerapan list

Deklarasi variabel dan data list adalah menggunakan *literal* list dengan notasi penulisan seperti berikut:

```
list_1 = [10, 70, 20]

list_2 = [
    'ab',
    'cd',
    'hi',
    'ca'
]

list_3 = [3.14, 'hello python', True, False]
```

Data dalam list biasa disebut dengan **element**. Setiap elemen disimpan dalam

list secara urut dengan penanda urutan yang disebut **index**. Nilai index dimulai dari angka `0`.

Sebagai contoh, pada variabel `list_1` di atas:

- Element index ke-`0` adalah data `10`
- Element index ke-`1` adalah data `70`
- Element index ke-`2` adalah data `20`

A.12.2. Perulangan data list

List adalah salah satu tipe data yang dapat digunakan langsung pada perulangan `for`. Contoh:

```
list_1 = [10, 70, 20]

for e in list_1:
    print("elem:", e)
```

Selain itu, perulangan list bisa juga dilakukan menggunakan index, contohnya seperti berikut:

```
list_1 = [10, 70, 20]
for i in range(0, len(list_1)):
    print("index:", i, "elem:", list_1[i])
```

Fungsi `len()` digunakan untuk menghitung jumlah element list. Dengan mengkombinasikan nilai balik fungsi ini dan fungsi `range()` bisa terbentuk data range dengan lebar sama dengan lebar list.

Lebih detailnya mengenai fungsi `len()` dibahas setelah ini

A.12.3. Operasi pada list

● Mengakses element via index

Nilai elemen list bisa diakses menggunakan notasi `list[index]`. Contoh:

```
list_1 = [10, 70, 20]

elem_1st = list_1[0]
elem_2nd = list_1[1]
elem_3rd = list_1[2]

print(elem_1st, elem_2nd, elem_3rd)
# output → [10, 70, 20]
```

● Slicing list

Slicing adalah metode pengaksesan list menggunakan notasi slice. Notasi ini mirip seperti array, namun mengembalikan data bertipe tetap slice.

Contoh pengaplikasian metode slicing bisa dilihat pada kode berikut. Variabel `list_2` diakses element-nya mulai index `1` hingga sebelum `3`:

```
list_2 = ['ab', 'cd', 'hi', 'ca']
print('list_2:', list_2)
# output → list2: ['ab', 'cd', 'hi', 'ca']

slice_1 = list_2[1:3]
```


Lebih detailnya mengenai slice dibahas pada chapter *Slice*

● Mengubah nilai element

Cara mengubah nilai element list dengan cara mengakses nilai element menggunakan index, kemudian diikuti operator assignment `=` dan nilai baru.

```
list_2 = ['ab', 'cd', 'hi', 'ca']
print('before:', list_2)
# output → before: ['ab', 'cd', 'hi', 'ca']

list_2[1] = 'zk'
list_2[2] = 'sa'
print('after: ', list_2)
# output → after: ['ab', 'zk', 'sa', 'ca']
```

● Append element

Operasi *append* atau menambahkan element baru setelah index terakhir, bisa menggunakan 2 cara:

- via method `append()`:

```
list_1 = [10, 70, 20]
print('before: ', list_1)
# output → before: [10, 70, 20]

list_1.append(88)
list_1.append(87)
print('after: ', list_1)
# output → after : [10, 70, 20, 88, 87]
```

- via slicing:

```
list_1 = [10, 70, 20]
print('before: ', list_1)
# output → before: [10, 70, 20]

list_1[len(list_1):] = [88, 87]
print('after: ', list_1)
# output → after : [10, 70, 20, 88, 87]
```

- Lebih detailnya mengenai method dibahas pada chapter *Method*

● Extend/concat element

Operasi *extend* (atau *concat*) mirip seperti *append*, perbedaannya ada pada data baru yang akan ditambahkan yang harus bertipe data list juga.

- via method `extend()`:

```
list_1 = [10, 70, 20]
list_2 = [88, 77]
list_1.extend(list_2)
print(list_1)
# output → [10, 70, 20, 88, 87]
```

- via slicing:

```
list_1 = [10, 70, 20]
list_2 = [88, 77]
list_1[len(list_1):] = list_2
```

- via operator `+`:

```
list_1 = [10, 70, 20]
list_2 = [88, 77]
list_3 = list_1 + list_2
print(list_3)
# output → [10, 70, 20, 88, 87]
```

Metode extend menggunakan operator `+` mengharuskan hasil operasi untuk ditampung ke variabel.

● Menyisipkan element pada index `i`

Method `insert()` digunakan untuk menyisipkan element baru pada posisi index tertentu (misalnya index `i`). Hasil operasi ini membuat semua element setelah index tersebut posisinya bergeser ke kanan.

Pada penggunaannya, para parameter pertama diisi dengan posisi index, dan parameter ke-2 diisi nilai.

```
list_3 = [10, 70, 20, 70]

list_3.insert(0, 15)
print(list_3)
# output → [15, 10, 70, 20, 70]

list_3.insert(2, 25)
print(list_3)
# output → [15, 10, 25, 70, 20, 70]
```

- Variabel `list_3` awalnya berisi `[10, 70, 20, 70]`

- Ditambahkan angka `15` pada index `0`, hasilnya nilai `list_3` sekarang adalah `[15, 10, 70, 20, 70]`
- Ditambahkan lagi, angka `25` pada index `2`, hasilnya nilai `list_3` sekarang adalah `[15, 10, 25, 70, 20, 70]`

● Menghapus element

Method `remove()` digunakan untuk menghapus element. Isi parameter fungsi dengan element yang ingin di hapus.

Jika element yang ingin dihapus ditemukan ada lebih dari 1, maka yang dihapus hanya yang pertama (sesuai urutan index).

```
list_3 = [10, 70, 20, 70]
```

```
list_3.remove(70)
print(list_3)
# output → [10, 20, 70]
```

```
list_3.remove(70)
print(list_3)
# output → [10, 20]
```

● Menghapus element pada index `i`

Method `pop()` berfungsi untuk menghapus element pada index tertentu. Jika tidak ada index yang ditentukan, maka data element terakhir yang dihapus.

Method `pop()` mengembalikan data element yang berhasil dihapus.

```
list_3 = [10, 70, 20, 70]
```

Jika index `i` yang ingin dihapus tidak ditemukan, maka error `IndexError` muncul.

```
list_3 = [10, 70, 20, 70]
x = list_3.pop(7)
```

```
⊗ list_3 = [10, 70, 20, 70] ...

-----
IndexError                                Traceback (most recent call last)
d:\Labs\Adam Studio\Ebook\dasarpemrogramanpython\dasarpemrogramanpython
  83 # %% A.12.2. ● Menghapus element pada index `i`
  84 list_3 = [10, 70, 20, 70]
----> 86 x = list_3.pop(7)
      87 print('list_3:', list_3)
      88 print('removed element:', x)

IndexError: pop index out of range
```

- Lebih detailnya mengenai error dibahas pada chapter *Error*

● Menghapus element pada range index

Python memiliki keyword `del` yang berguna untuk menghapus suatu data. Dengan menggabungkan keyword ini dan operasi slicing, kita bisa menghapus element dalam range tertentu dengan cukup mudah.

Contoh, menghapus element pada index `1` hingga sebelum `3`:

```
list_3 = [10, 70, 20, 70]

del list_3[1:3]
```

● Menghitung jumlah element

Fungsi `len()` digunakan untuk menghitung jumlah element.

```
list_3 = [10, 70, 20, 70]
total = len(list_3)
print(total)
# output → 4
```

Selain fungsi `len()`, ada juga method `count()` milik method slice yang kegunaannya memiliki kemiripan. Perbedaannya, method `count()` melakukan operasi pencarian sekaligus menghitung jumlah element yang ditemukan.

Agar lebih jelas, silakan lihat kode berikut:

```
list_3 = [10, 70, 20, 70]
count = list_3.count(70)
print('jumlah element dengan data `70`: ', count)
# output → jumlah element dengan data `70`: 2
```

● Mencari index element list

Untuk mencari index menggunakan nilai element, gunakan method `index()` milik list. Contoh bisa dilihat berikut, data `cd` ada dalam list pada index `1`.

```
list_2 = ['ab', 'cd', 'hi', 'ca']

idx_1st = list_2.index('cd')
print('idx_1st: ', idx_1st)
# output → idx_1st: 1
```

Jika data element yang dicari tidak ada, maka akan muncul error `ValueError` :

```
idx_2nd = list_2.index('kk')
print('idx_2nd: ', idx_2nd)
```

```
⊗ idx_2nd = list_2.index('kk') ...

-----
ValueError                                Traceback (most recent call last)
d:\Labs\Adam Studio\Ebook\dasarpemrogramanpython\dasarpemrogramanpython\examples
  36 # %%
----> 37 idx_2nd = list_2.index('kk')
      38 print('idx_2nd: ', idx_2nd)

ValueError: 'kk' is not in list
```

● Mengosongkan list

Ada dua cara untuk mengosongkan list:

- via method `clear()` :

```
list_1 = [10, 70, 20]
list_1.clear()
print(list_1)
# output → []
```

- Menimpanya dengan `[]` :

```
list_1 = [10, 70, 20]
list_1 = []
print(list_1)
# output → []
```

- Menggunakan keyword `del` dan slicing:

```
list_1 = [10, 70, 20]
del list_1[:]
print(list_1)
# output → []
```

● Membalik urutan element list

Method `reverse()` digunakan untuk membalik posisi element pada list.

```
list_1 = [10, 70, 20]
list_1.reverse()
print(list_1)
# output → [20, 70, 10]
```

● Cloning/duplikasi list

Ada 2 cara untuk menduplikasi list:

- Menggunakan method `copy()`:

```
list_1 = [10, 70, 20]
list_2 = list_1.copy()
print(list_1)
# output → [10, 70, 20]
print(list_2)
# output → [10, 70, 20]
```

- Kombinasi operasi assignment dan slicing:


```
list_1 = [10, 70, 20]
list_2 = list_1[:]
print(list_1)
# output → [10, 70, 20]
print(list_2)
# output → [10, 70, 20]
```

● Sorting

Mengurutkan data list bisa dilakukan menggunakan *default sorter* milik Python, yaitu method `sort()`.

```
list_1 = [10, 70, 20]
list_1.sort()
print(list_1)
# output → [10, 20, 70]

list_2 = ['z', 'h', 'c']
list_2.sort()
print(list_2)
# output → ['c', 'h', 'z']
```

```
▼ TERMINAL
PS D:\labs> python.exe main.py
[10, 20, 70]
['c', 'h', 'z']
```

Method ini sebenarnya menyidakan kapasitas sorting yang cukup advance, caranya dengan cara menambahkan closure/lambda pada argument method ini.

Lebih detailnya mengenai closure/lambda dibahas pada chapter [Closure/](#)

A.12.4. Nested list

Penulisan nested list cukup mudah, contohnya bisa dilihat pada program matrix berikut:

```
matrix = [  
    [0, 1, 0, 1, 0],  
    [1, 1, 1, 0, 0],  
    [0, 0, 0, 1, 1],  
    [0, 1, 1, 1, 0],  
]  
  
for row in matrix:  
    for cel in row:  
        print(cel, end=" ")  
    print()
```

```
▼ TERMINAL  
PS D:\labs> python.exe main.py  
0 1 0 1 0  
1 1 1 0 0  
0 0 0 1 1  
0 1 1 1 0
```

Catatan chapter

● Source code praktik

github.com/novalagung/dasarpemrogramanpython-example/..list

● Chapter relevan lainnya

- List Comprehension
- Perulangan → for, range
- Slice
- Closure/lambda

● Referensi

- <https://docs.python.org/3/tutorial/datastructures.html>

A.13. List Comprehension

List comprehension adalah metode ringkas pembuatan list selain menggunakan literal `[]`. Cara ini lebih banyak diterapkan untuk operasi list yang menghasilkan struktur baru.

Pada chapter ini kita akan mempelajarinya.

A.13.1. Penerapan list comprehension

Metode penulisan ini membuat kode menjadi sangat ringkas, dengan konsekuensi agak sedikit membingungkan untuk yang belum terbiasa. Jadi gunakan sesuai kebutuhan.

Silakan pelajari contoh berikut agar lebih mudah memahami seperti apa itu *list comprehension*.

🕒 Contoh #1

Perulangan berikut:

```
seq = []  
for i in range(5):  
    seq.append(i * 2)
```

... bisa dituliskan lebih ringkas menggunakan *list comprehension*, menjadi seperti berikut:

```
seq = [i * 2 for i in range(5)]  
  
print(seq) # output → [0, 2, 4, 6, 8]
```

● Contoh #2

Perulangan berikut:

```
seq = []  
for i in range(10):  
    if i % 2 == 1:  
        seq.append(i)  
  
print(seq) # output → [1, 3, 5, 7, 9]
```

... bisa dituliskan lebih ringkas menggunakan *list comprehension*, menjadi seperti berikut:

```
seq = [i for i in range(10) if i % 2 == 1]  
  
print(seq) # output → [1, 3, 5, 7, 9]
```

● Contoh #3

Perulangan berikut:

```
seq = []
```

... bisa dituliskan lebih ringkas menjadi dengan bantuan *ternary* menjadi seperti ini:

```
seq = []
for i in range(1, 10):
    seq.append(i * (2 if i % 2 == 0 else 3))

print(seq) # output → [3, 4, 9, 8, 15, 12, 21, 16, 27]
```

... dan bisa dijadikan lebih ringkas lagi menggunakan *list comprehension*:

```
seq = [(i * 2) if i % 2 == 0 else (i * 3) for i in range(1, 10)]

print(seq) # output → [3, 4, 9, 8, 15, 12, 21, 16, 27]
```

🕒 Contoh #4

Perulangan berikut:

```
list_x = ['a', 'b', 'c']
list_y = ['1', '2', '3']

comb = []
for x in list_x:
    for y in list_y:
        comb.append(x + y)

print(seq) # output → ['a1', 'a2', 'a3', 'b1', 'b2', 'b3', 'c1', 'c2', 'c3']
```

... bisa dituliskan lebih ringkas menggunakan *list comprehension*, menjadi seperti berikut:

```
comb = [x + y for x in list_x for y in list_y]

print(seq) # output → ['a1', 'a2', 'a3', 'b1', 'b2', 'b3', 'c1',
                       'c2', 'c3']
```

● Contoh #5

Perulangan berikut:

```
matrix = [
    [1, 2, 3, 4],
    [5, 6, 7, 8],
    [9, 10, 11, 12],
]

transposed = []
for i in range(4):
    tr = []
    for row in matrix:
        tr.append(row[i])
    transposed.append(tr)

print(transposed) # output → [[1, 5, 9], [2, 6, 10], [3, 7, 11],
                              [4, 8, 12]]
```

... bisa dituliskan lebih ringkas menjadi seperti ini:

```
matrix = [
    [1, 2, 3, 4],
    [5, 6, 7, 8],
    [9, 10, 11, 12],
]
```

... dan bisa dijadikan lebih ringkas lagi menggunakan *list comprehension*:

```
matrix = [  
    [1, 2, 3, 4],  
    [5, 6, 7, 8],  
    [9, 10, 11, 12],  
]  
  
transposed = [[row[i] for row in matrix] for i in range(4)]  
  
print(transposed) # output → [[1, 5, 9], [2, 6, 10], [3, 7, 11],  
[4, 8, 12]]
```

Catatan chapter

● Source code praktik

github.com/novalagung/dasarpemrogramanpython-example/..../list-comprehension

● Chapter relevan lainnya

- List
- Perulangan → for, range

● TBA

- Stack vs Queue

● Referensi

- <https://docs.python.org/3/tutorial/datastructures.html>