



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 3.11.3**.

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:

[Dasar Pemrograman Python.pdf](#)

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

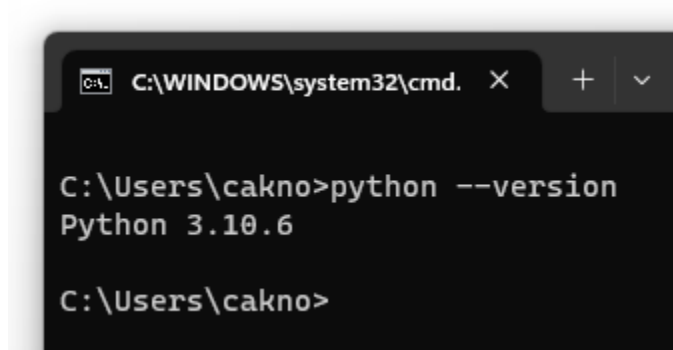
- Tarball source code bisa diunduh di [situs official Python](#)

● 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:

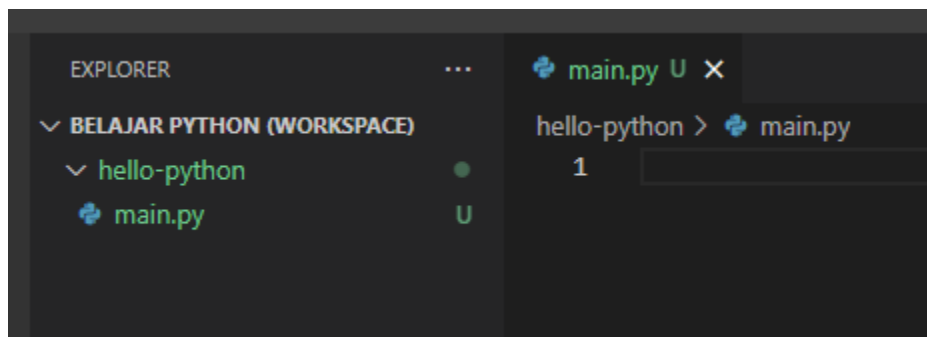
- Extension [Python](#), untuk mendapatkan benefit API doc, autocompletion, linting, run feature, dan lainnya.
- Extension [Jupyter](#), untuk interactive run program via editor.

A.1. Python Hello World

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>  
python main.py
```

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")
```

Below the code editor is a 'TERMINAL' panel showing the command prompt output:

```
PS D:\Labs\hello-python> python main.py
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()`). 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://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

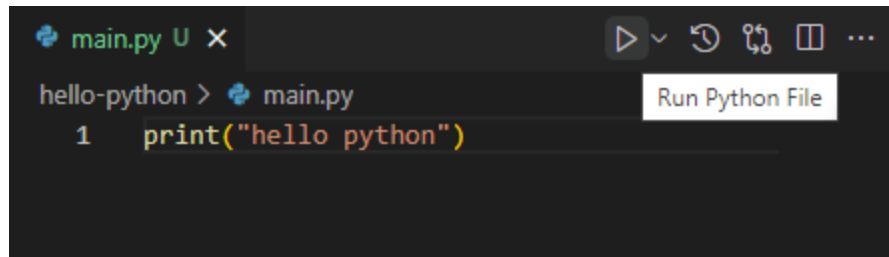
● Menggunakan command `python`

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

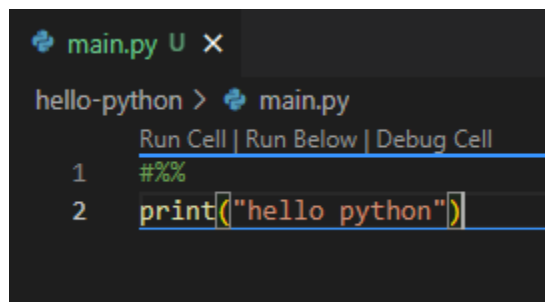
Cara run program ini lebih praktis karena tinggal klik-klik saja. Di toolbar VSCode sebelah kanan atas ada tombol , gunakan tombol tersebut untuk menjalankan program.



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

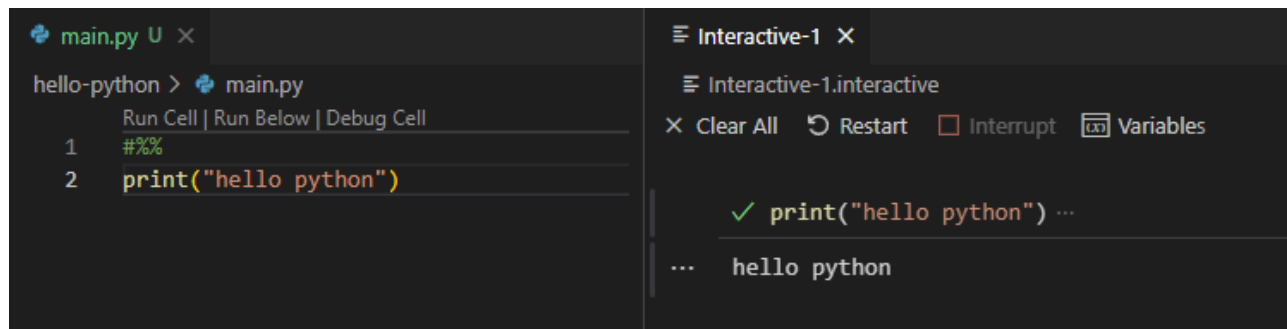
● Menggunakan jupyter code cells

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



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

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



```
main.py U x
hello-python > main.py
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. Python 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.

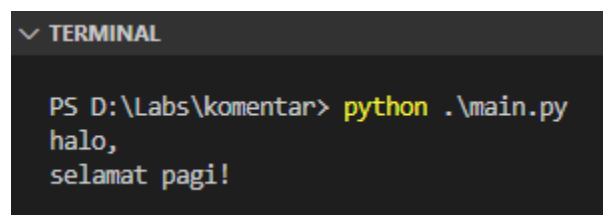
Python mengenal dua jenis komentar, yaitu komentar satu baris dan multi-baris.

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 `#` dituliskan

```
# 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 & Operasi String*

Catatan chapter

● Source code praktik

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

● Chapter relevan lainnya

- [String & Operasi String](#)

● Referensi

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

A.4. Python 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
```

Karakter `=` adalah **operator assignment**, digunakan untuk operasi penugasan. Nilai yang ada di sebelah kanan `=` ditugaskan untuk ditampung oleh variabel yang berada di sebelah kiri `=`. Contoh pada statement `nama = "noval"`, nilai `"noval"` ditugaskan untuk ditampung oleh variabel `nama`.

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

Ok. Selanjutnya, coba kita munculkan nilai ke-empat variabel di atas ke layar

menggunakan fungsi `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
```

Penjelasan mengenai program di atas bisa dilihat di bawah ini:

● **Output formatting** `print`

Di program yang sudah ditulis, ada statement berikut:

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

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

Lalu di bawahnya ada statement ini, yang merupakan contoh 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`. Contoh penerapannya bisa dilihat pada statement ke-3 program yang sudah di tulis.

```
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

Di pemrograman Python, deklarasi variabel adalah pasti operasi assignment. Variabel dideklarasikan dengan ditentukan langsung nilai awalnya.

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

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

● Chapter relevan lainnya

- Tipe Data
- Output Formatting

● 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. Python Konstanta

Konstanta (atau nilai konstan) adalah sebuah variabel yang nilainya didefinisikan di awal dan tidak bisa diubah setelahnya.

Pada chapter ini kita akan mempelajari tentang penerapan Konstanta di Python.

A.5.1. Konstanta di Python

Deklarasi konstanta di Python dilakukan menggunakan bantuan tipe *class* bernama `typing.Final`.

Untuk menggunakannya, `typing.Final` perlu di-import terlebih dahulu menggunakan keyword `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 boleh tidak ditentukan (tipe akan diidentifikasi oleh interpreter berdasarkan tipe data nilainya).

```
# 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 Data](#)

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

● Chapter relevan lainnya

- [Module Import](#)
- [Python standard library \(stdlib\)](#)

● Referensi

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

A.6. Python 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* 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>

Lebih detailnya mengenai string dibahas pada chapter *Numeric*

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*

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, dengan isi bisa berupa 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]
```

Lebih detailnya mengenai list dibahas pada chapter [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)
print(tuple_1[2])
# output → 4
```

Lebih detailnya mengenai tuple dibahas pada chapter [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 *Dictionary*

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 [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

● Chapter relevan lainnya

- String & Operasi String
- List
- Tuple
- Dictionary
- Sets

● Referensi

- <https://docs.python.org/3/tutorial/introduction.html>
 - <https://docs.python.org/3/library/stdtypes.html#typeseq>
-

A.7. Python 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 (penugasan nilai atau penentuan nilai), sekaligus untuk deklarasi variabel jika variabel tersebut sebelumnya belum terdeklarasi. Contoh:

```
# deklarasi variabel num_1
num_1 = 12

# deklarasi variabel num_2
num_2 = 24

# nilai baru ditugaskan ke variabel num_2
num_2 = 12

# deklarasi variabel num_3 dengan isi nilai hasil operasi aritmatika
`num_1 + num_2`
```

A.7.3. Operator perbandingan

Operator perbandingan pasti menghasilkan nilai kebenaran `bool` dengan kemungkinannya 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 karakter spasi () 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 `id()` kemudian tulis data yang ingin diambil ID-nya sebagai argument pemanggilan fungsi tersebut.

```
data_1 = "hello world"
```

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

sample_set = { "sesuk", "preiiii" }
is_prei = "preiiii" in sample_set
print(is_prei)
# True

sample_str = 'Hello world'
is_substring_exists = 'orl' in sample_str
print(is_substring_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

● Chapter relevan lainnya

- Variabel
- Tipe Data
- Output Formatting

● Referensi

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

A.8. Seleksi kondisi

Python → if, elif, else

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

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

A.8.1. Keyword `if`

`if` adalah keyword seleksi kondisi di Python. Cara penerapan keyword ini sangat mudah, cukup tulis saja `if` 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 (masih satu *chain*). Contoh:

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

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 praktikan kode berikut:

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

```
✓ TERMINAL

PS D:\Labs\if-elif-else> python.exe main.py
Enter your grade: 78
inputan 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 ditampung 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:
    print("passed the exam")

    if grade <= 70:
        print("but you need to improve it!")
    else:
        print("with ok grade")

else:
    print("below the passing grade")
```

▼ 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 sederhana berikut, isinya adalah seleksi kondisi sederhana pengecekan nilai `grade >= 65` atau tidak.


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

Kode di atas bisa dituliskan dalam bentuk alternatif penulisan kode lainnya:

● **One-line / sebaris**

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

Metode penulisan sebaris ini cocok diterapkan pada situasi dimana seleksi kondisi hanya memiliki 1 kondisi saja.

● **Ternary**

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

Metode penulisan *ternary* umum 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 sebenarnya adalah sama seperti penerapan ternary sebelumnya, perbedaannya: pada metode ini setiap kondisi menghasilkan nilai

balik yang umumnya ditampung oleh variabel. Pada contoh di atas, nilai balik ditampung 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 Python → for, range

Perulangan atau *loop* merupakan teknik untuk mengulang-ulang eksekusi suatu blok kode, atau mengiterasi elemen milik tipe data kolektif (contohnya: list). Chapter ini membahas tentang penerapannya di Python.

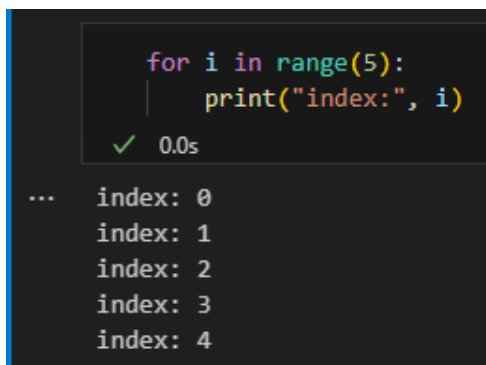
A.9.1. Keyword `for` dan fungsi `range()`

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
```

Penjelasan:

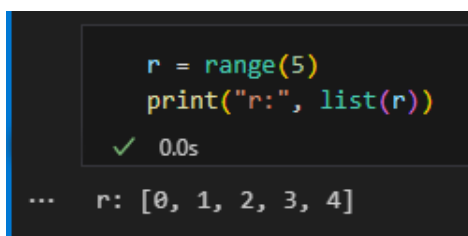
- Statement `print("index:", i)` muncul 5 kali, karena perulangan dilakukan dengan kontrol `range(5)` dimana statement tersebut menghasilkan object *range* dengan isi deret angka sejumlah 5 dimulai dari angka 0 hingga 4.
- Statement `for i in range(5):` adalah contoh penulisan perulangan menggunakan `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 jenis data yang strukturnya mirip seperti list (tapi bukan list) yang kegunaan utamanya adalah untuk kontrol perulangan.

Object *sequence* bisa dikonversi bentuk list dengan cara dibungkus menggunakan fungsi `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

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
```

```
for i in range(0, 3, 1):  
    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(5, -5, -1):  
    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)  
✓ 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. Iterasi element data kolektif

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 chapter **Fungsi***

● Fungsi `print()` tanpa parameter

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

Catatan chapter

● Source code praktik

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

● 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 Python

→ 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 *sequence*, contohnya seperti range dan list.

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

● Referensi

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

A.11. Perulangan Python

→ 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 (karena 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 program berikut, yang spesifikasinya:

- Program berisi perulangan dengan kontrol adalah data *range* sebanyak 10 (dimana isinya adalah angka numerik `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()
```

Penjelasan:

- 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. Python 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:

```
# contoh list
list_1 = [10, 70, 20]

# list dengan deklarasi element secara vertikal
list_2 = [
    'ab',
    'cd',
    'hi',
    'ca'
]

# list dengan element berisi bermacam-macam tipe data
list_3 = [3.14, 'hello python', True, False]

# list kosong
list_4 = []
```

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 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

● Fungsi `enumerate()`

Fungsi `enumerate()` digunakan untuk membuat data sequence menjadi data enumerasi, yang jika dimasukkan ke perulangan di setiap iterasinya bisa kita akses index beserta element-nya.

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

for i, v in enumerate(list_1):
    print("index:", i, "elem:", v)
```

A.12.3. 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
```

A.12.4. Fungsi `list()`

● Konversi range ke list

Data range (hasil pemanggilan fungsi `range()`) bisa dikonversi ke bentuk list menggunakan fungsi `list()`. Cara ini cukup efisien untuk pembuatan data list yang memiliki *pattern* atau pola. Sebagai contoh:

- List dimulai angka `0` hingga `9`:

```
range_1 = range(0, 10)
list_1 = list(range_1)
print(list_1)
# output → [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

- List dimulai angka `1` hingga `21` dengan penambahan `3`:

```
range_2 = range(0, 22, 3)
list_2 = list(range_2)
print(list_2)
# output → [0, 3, 6, 9, 12, 15, 18, 21]
```

- List dimulai angka `100` hingga `0` dengan pengurangan `-10`:

```
range_3 = range(100, 0, -10)
list_3 = list(range_3)
print(list_3)
# output → [100, 90, 80, 70, 60, 50, 40, 30, 20, 10]
```

Selain metode ini, ada juga cara lainnya untuk membuat list, yaitu menggunakan metode list comprehension, yang akan dibahas pada chapter berikutnya, yaitu *List Comprehension*

● Konversi string ke list

Selain untuk konversi data range ke list, fungsi `list()` bisa digunakan untuk konversi data string ke list, dengan hasil adalah setiap karakter string menjadi element list.

```
alphabets = list('abcdefgh')
print(alphabets)
# output → ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h']
```

● Konversi tuple ke list

Tipe data tuple bisa diubah bentuknya menjadi list dengan menggunakan fungsi `list()`. Contoh penerapannya:

```
tuple_1 = (1, 2, 3, 4)
numbers = list(tuple_1)
print(numbers)
# output → [1, 2, 3, 4]
```

Lebih detailnya mengenai tuple dibahas pada chapter *Tuple*

A.12.5. 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]
```

DANGER

Pengaksesan elemen menggunakan index di-luar kapasitas data akan menghasilkan error.

Sebagai contoh, data `list_1` di atas jika diakses index ke-3-nya misalnya (`list_1[3]`) hasilnya adalah error.

● Mengecek apakah element ada

Kombinasi keyword `if` dan `in` bisa digunakan untuk mengidentifikasi apakah suatu element merupakan bagian dari list atau tidak. Contoh penerapannya:

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

if n in list_1:
    print(n, "is exists")
else:
    print(n, "is NOT exists")

# output → 70 is exists
```

● **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]
print('slice_1:', slice_1)
# output → slice_1: ['cd', 'hi']
```

*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/union element**

Operasi *extend* (atau *concat* atau *union*) adalah operasi penggabungan dua data list. Ada beberapa metode yang tersedia, diantaranya:

- 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
print(list_1)
# output → [10, 70, 20, 88, 87]
```

- 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]
```

```
x = list_3.pop(2)
print('list_3:', list_3)
# output → list_3: [10, 70, 70]
print('removed element:', x)
# output → removed element: 20
```

```
x = list_3.pop()
print('list_3:', list_3)
# output → list_3: [10, 70]
print('removed element:', x)
# output → removed element: 70
```

Jika index **i** yang ingin dihapus tidak diketemukan, 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*

Selain menggunakan method `pop()`, keyword `del` bisa difungsikan untuk hal yang sama, yaitu menghapus elemen tertentu. Contoh penerapannya:

```
list_3 = [10, 70, 20, 70]
print('len:', len(list_3), "data:", list_3)

del list_3[1]
print('len:', len(list_3), "data:", list_3)
```

● 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]
print(list_3)
# output → [10, 70]
```

● 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]
```

● Copy list

Ada 2 cara untuk menduplikasi list, menggunakan method `copy()` dan teknik

slicing.

- 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]
```

Operasi copy disini jenisnya adalah shallow copy.

Lebih detailnya mengenai shallow copy vs deep copy dibahas pada chapter terpisah.

● Sorting

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

```
list_1 = [10, 70, 20]
list_1.sort()
```

```
▼ 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/Lambda](#)

Catatan chapter

● Source code praktik

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

● Chapter relevan lainnya

- [List Comprehension](#)
- [Perulangan → for, range](#)
- [Slice](#)
- [Closure/lambda](#)

● TBA

- Pack & Unpack with `*` & `**`

● Referensi

- <https://docs.python.org/3/tutorial/datastructures.html>
 - <https://docs.python.org/3/library/stdtypes.html#typeseq>
-

A.13. Python List Comprehension

List comprehension adalah metode ringkas pembuatan list (selain menggunakan literal `[]` atau menggunakan fungsi `list()`). 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 list comprehension membuat kode menjadi sangat ringkas, dengan konsekuensi agak sedikit membingungkan untuk yang belum terbiasa. Jadi penulis sarankan 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)

print(seq) # output → [0, 2, 4, 6, 8]
```

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

... 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 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 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],
]

transposed = []
```

... 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>
 - <https://docs.python.org/3/library/stdtypes.html#typeseq>
-

A.14. Python Tuple

Tuple adalah tipe data sequence yang ideal digunakan untuk menampung nilai kolektif yang isinya tidak akan berubah (*immutable*), berbeda dengan list yang lebih cocok untuk data yang bisa berubah nilai elemen-nya (*mutable*).

Pada chapter ini kita akan belajar tentang topik ini.

A.15.1. Tuple vs. List

Tipe data tuple sekilas memiliki beberapa kemiripan dan juga perbedaan jika dibandingkan dengan list.

	Tuple	List
Literal	<code>()</code> , atau <code>tuple()</code> , atau elemen ditulis tanpa <code>()</code>	<code>[]</code> , atau <code>list()</code>
Contoh	<pre>x = ()</pre> <pre>x = tuple()</pre> <pre>x = (1, True, "h", 2, 1)</pre> <pre>x = 1, True, "h", 2, 1</pre>	<pre>x = []</pre> <pre>x = list()</pre> <pre>x = [1, True, "h", 2, 1]</pre>
Urutan elemen	urut sesuai index	

	Tuple	List
Pengaksesan elemen	via index dan perulangan	
<i>Mutability</i>	elemen tidak bisa diubah	elemen bisa diubah
Duplikasi elemen	elemen bisa duplikat	
Tipe data elemen	bisa sejenis maupun berbeda satu sama lain	

A.14.2. Penerapan tuple

Deklarasi tuple menggunakan literal `()` dengan delimiter tanda koma `,`. Contoh syntax-nya bisa dilihat pada kode berikut:

```
tuple_1 = (2, 3, 4, "hello python", False)

print("data:", tuple_1)
# output → data: (2, 3, 4, "hello python", False)

print("total elem:", len(tuple_1))
# output → total elem: 5
```

- Tuple bisa menampung element yang tipe datanya bisa sejenis bisa tidak, sama seperti list.
- Fungsi `len()` digunakan untuk menghitung lebar tuple.

A.14.3. Mengakses element tuple via index

Element tuple bisa diakses menggunakan notasi `tuple[index]`.

```
tuple_1 = (2, 3, 4, 5)

print("elem 0:", tuple_1[0])
# output → elem 0: 2

print("elem 1:", tuple_1[1])
# output → elem 1: 3
```

DANGER

Pengaksesan elemen menggunakan index di-luar kapasitas data akan menghasilkan error.

Sebagai contoh, data `tuple_1` di atas jika diakses index ke-4-nya misalnya (`tuple_1[4]`) hasilnya adalah error.

A.14.4. Perulangan tuple

Tuple adalah salah satu tipe data yang bisa digunakan secara langsung pada perulangan menggunakan keyword `for`.

Pada contoh berikut, variabel `tuple_2` dimasukan ke blok perulangan. Di setiap iterasinya, variabel `t` berisi element tuple.

```
tuple_2 = ('ultra instinc shaggy', 'nightwing', 'noob saibot')

for t in tuple_2:
    print(t)
```



```
ultra instinc shaggy
nightwing
noob saibot
```

Perulangan di atas ekuivalen dengan perulangan berikut:

```
tuple_2 = ('ultra instinc shaggy', 'nightwing', 'noob saibot')

for i in range(0, len(tuple_2)):
    print("index:", i, "elem:", tuple_2[i])
```

● Fungsi `enumerate()`

Fungsi `enumerate()` digunakan untuk membuat data sequence menjadi data enumerasi, yang jika dimasukkan ke perulangan di setiap iterasinya bisa kita akses index beserta element-nya.

```
tuple_2 = ('ultra instinc shaggy', 'nightwing', 'noob saibot')

for i, v in enumerate(tuple_2):
    print("index:", i, "elem:", v)
```

A.14.5. Mengecek apakah element ada

Kombinasi keyword `if` dan `in` bisa digunakan untuk mengidentifikasi apakah suatu element merupakan bagian dari tuple atau tidak. Contoh penerapannya:

```
tuple_1 = (10, 70, 20)
n = 70

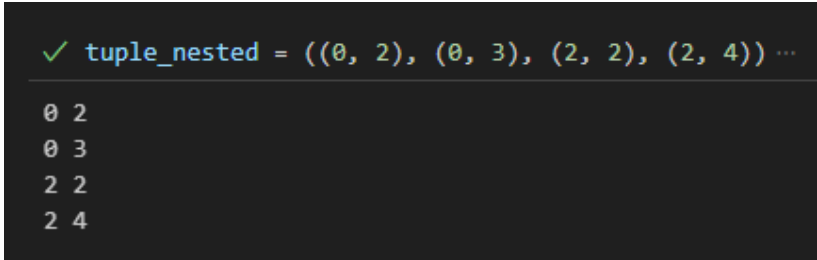
if n in tuple_1:
    print(n, "is exists")
else:
```

A.14.6. Nested tuple

Nested tuple dibuat dengan menuliskan data tuple sebagai element tuple. Contoh:

```
tuple_nested = ((0, 2), (0, 3), (2, 2), (2, 4))

for row in tuple_nested:
    for cell in row:
        print(cell, end=" ")
    print()
```



```
✓ tuple_nested = ((0, 2), (0, 3), (2, 2), (2, 4)) ...
0 2
0 3
2 2
2 4
```

Penulisan data literal nested tuple bisa dalam bentuk horizontal maupun vertikal. Perbandingannya bisa dilihat pada kode berikut:

```
# horizontal
tuple_nested = ((0, 2), (0, 3), (2, 2), (2, 4))

# vertikal
tuple_nested = (
    (0, 2),
    (0, 3),
    (2, 2),
    (2, 4)
)
```

A.14.7. List dan tuple

Tipe data list dan tuple umum dikombinasikan. Keduanya sangat mirip tapi memiliki perbedaan yang jelas, yaitu nilai tuple tidak bisa dimodifikasi sedangkan list bisa.

```
# deklarasi data list berisi elemen tuple
data = [
    ("ultra instinc shaggy", 1, True, ['detective', 'saiyan']),
    ("nightwing", 3, True, ['teen titans', 'bat family']),
]

# append tuple ke list
data.append(("noob saibot", 6, False, ['brotherhood of shadow']))

# append tuple ke list
data.append(("tifa lockhart", 2, True, ['avalanche']))

# print data
print("name | rank | win | affiliation")
print("-----")
for row in data:
    for cell in row:
        print(cell, end=" | ")
    print()
```

```
name | rank | win | affiliation
-----
ultra instinc shaggy | 1 | True | ['detective', 'saiyan'] |
nightwing | 3 | True | ['teen titans', 'bat family'] |
noob saibot | 6 | False | ['brotherhood of shadow'] |
tifa lockhart | 2 | True | ['avalanche'] |
```

A.14.8. Fungsi `tuple()`

● Konversi string ke tuple

Fungsi `tuple()` bisa digunakan untuk konversi data string ke tuple. Hasilnya adalah nilai tuple dengan element berisi setiap karakter yang ada di string. Contoh:

```
alphabets = tuple('abcdefgh')
print(alphabets)
# output → ('a', 'b', 'c', 'd', 'e', 'f', 'g', 'h')
```

● Konversi list ke tuple

Konversi list ke tuple bisa dilakukan dengan mudah menggunakan fungsi `tuple()`. Contoh penerapannya:

```
numbers = tuple([2, 3, 4, 5])
print(numbers)
# output → (2, 3, 4, 5)
```

● Konversi range ke tuple

Range juga bisa dikonversi ke tuple menggunakan fungsi `tuple()`.

```
r = range(0, 3)
rtuple = tuple(r)
print(rtuple)
# output → (0, 1, 2)
```

A.14.9. Tuple *packing* dan *unpacking*

● Tuple *packing*

Packing adalah istilah untuk menggabungkan beberapa data menjadi satu data kolektif. Contoh pengaplikasiannya bisa dilihat pada program berikut, ada 3 variabel dengan isi berbeda di-*pack* menjadi satu data tuple.

```
first_name = "aerith gainsborough"
rank = 11
win = False

row_data = (first_name, rank, win)

print(row_data)
# output → ('aerith gainsborough', 11, False)
```

Bisa dilihat penerapan metode *packing* cukup mudah. Tulis saja data atau variabel yang ingin di-*pack* dalam notasi tuple, kemudian gunakan sebagai nilai pada operasi *assignment*.

Pada contoh di atas, variabel `row_data` menampung nilai tuple hasil *packing* variabel `first_name`, `rank`, dan `win`.

O iya, penulisan tuple boleh juga dituliskan tanpa menggunakan karakter `(` & `)`.

```
# dengan ()
row_data = (first_name, rank, win)

# tanpa ()
row_data = first_name, rank, win
```

Namun, pastikan untuk hati-hati dalam penerapan penulisan tuple tanpa `()`, karena bisa jadi salah paham. Jangan gunakan metode ini pada saat menggunakan

tuple sebagai nilai argument pemanggilan fungsi, karena interpreter akan menganggapnya sebagai banyak argument.

```
# fungsi print() dengan satu argument berisi tuple (first_name, rank, win)
print((first_name, rank, win))

# fungsi print() dengan isi 3 arguments: first_name, rank, win
print(first_name, rank, win)
```

● Tuple *unpacking*

Unpacking adalah istilah untuk menyebar isi suatu data kolektif ke beberapa variabel. *Unpacking* merupakan kebalikan dari *packing*.

Contoh penerapan tuple *unpacking*:

```
row_data = ('aerith gainsborough', 11, False)
first_name, rank, win = row_data

print(first_name, rank, win)
# output → aerith gainsborough 11 False
```

A.14.10. Tuple kosong `()`

Tuple bisa saja tidak berisi apapun, contohnya data `()`, yang cukup umum digunakan untuk merepresentasikan data kolektif yang isinya bisa saja kosong.

```
empty_tuple = ()
print(empty_tuple)
# output → ()
```

Berikut adalah contoh penerapannya, dimisalkan ada data kolektif yang didapat dari database berbentuk array object. Data tersebut perlu disimpan oleh variabel list

yang element-nya adalah tuple dengan spesifikasi:

- Tuple element index 0 berisi `name` .
- Tuple element index 1 berisi `rank` .
- Tuple element index 2 berisi `win` .
- Tuple element index 3 berisi `affliation` , dimana `affliation` bisa saja kosong.

Sample data bisa dilihat berikut ini:

```
data = [  
    ("ultra instinc shaggy", 1, True, ('detective', 'saiyan')),  
    ("nightwing", 3, True, ('teen titans', 'bat family')),  
    ("kucing meong", 7, False, ()),  
]
```

Bisa dilihat data `kucing meong` tidak memiliki `affliation` , karena terisi dengan nilai tuple `()` .

Catatan chapter

● Source code praktik

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

● Chapter relevan lainnya

- [List](#)

● TBA

- Pack & Unpack with `*` & `**`

- Zip

● Referensi

- <https://docs.python.org/3/tutorial/datastructures.html>
 - <https://docs.python.org/3/library/stdtypes.html#typeseq>
-

A.15. Python Set

Set adalah tipe data yang digunakan untuk menampung nilai kolektif unik, jadi tidak ada duplikasi elemen. Elemen yang ada pada set disimpan secara tidak urut.

Pada chapter ini, selain mempelajari tentang `set` kita akan bahas juga satu variasinya yaitu `frozenset`.

A.15.1. Set vs. Tuple vs. List

Tipe data set sekilas memiliki kemiripan jika dibandingkan dengan tuple dan list, namun sebenarnya lebih banyak perbedaannya. Silakan lihat tabel berikut untuk lebih jelasnya.

	Set	Tuple		List
Literal	<code>set()</code> , atau elemen ditulis diapit <code>{</code> dan <code>}</code>	<code>()</code> , atau <code>tuple()</code> , atau elemen ditulis tanpa <code>()</code>		<code>[]</code> , atau <code>list()</code>
Contoh	<code>x = set()</code>	<code>x = ()</code>	<code>x = []</code>	
	<code>x = {1, True, "h", 2}</code>	<code>x = tuple()</code>	<code>x = list()</code>	
		<code>x = (1, True, "h", 2, 1)</code>	<code>x = [1, True, "h", 2, 1]</code>	
		<code>x = 1, True, "h", 2, 1</code>		
Urutan elemen	tidak urut	urut sesuai index		
Pengaksesan elemen	hanya via perulangan	via index dan perulangan		
Mutability	elemen bisa diubah	elemen tidak bisa diubah	elemen bisa diubah	
Duplikasi elemen	elemen selalu unik	elemen bisa duplikat		
Tipe data elemen	bisa sejenis maupun berbeda satu sama lain			

A.15.2. Penerapan set

Implementasi tipe data set cukup mudah, langsung tulis saja nilai elemen dengan separator `,` dan diapit menggunakan tanda kurung kurawal `{ }`. Contoh:

```
data_1 = {1, 'abc', False, ('banana', 'spaghetti')}

print("data:", data_1)
# output → data: {1, 'abc', False, ('banana', 'spaghetti')}
```

```
print("len:", len(data_1))
# output → len: 3
```

- Set bisa menampung element yang tipe datanya bisa sejenis bisa tidak, sama seperti tuple dan list.
- Fungsi `len()` digunakan untuk menghitung lebar set.

! INFO

Untuk deklarasi set kosong (tanpa isi), gunakan fungsi `set()`, bukan `{}` karena literal tersebut akan menciptakan data bertipe lainnya yaitu dictionary.

```
data_2 = set()

print("data:", data_2)
# output → data: set()

print("len:", len(data_2))
# output → len: 0
```

Hanya gunakan kurung kurawal buka dan tutup untuk deklarasi set yang ada elemennya (tidak kosong).

A.15.3. Mengakses elemen set

Nilai set *by default* hanya bisa diakses menggunakan perulangan:

```
fellowship = {'aragorn', 'gimli', 'legolas'}

for p in fellowship:
    print(p)
```

```
legolas
aragorn
gimli
```

Dari limitasi ini, set difungsikan untuk menyelesaikan masalah yang cukup spesifik seperti eliminasi elemen

duplikat.

● Eliminasi elemen duplikat

Tipe data set memang didesain untuk menyimpan data unik, duplikasi elemen tidak mungkin terjadi, bahkan meskipun dipaksa. Contoh:

```
data = {1, 2, 3, 2, 1, 4, 5, 2, 3, 5}
print(data)
# output → {1, 2, 3, 4, 5}
```

Variabel `data` yang diisi dengan data set dengan banyak elemen duplikasi, sewaktu di-print elemennya adalah unik.

Ok, selanjutnya, pada contoh kedua berikut kita akan coba gunakan set untuk mengeliminasi elemen duplikat pada suatu list.

```
data = [1, 2, 3, 2, 1, 4, 5, 2, 3, 5]
print(data)
# output → [1, 2, 3, 2, 1, 4, 5, 2, 3, 5]

data_unique_set = set(data)
print(data_unique_set)
# output → {1, 2, 3, 4, 5}

data_unique = list(data_unique_set)
print(data_unique)
# output → [1, 2, 3, 4, 5]
```

Penjelasan untuk kode di atas:

- Variabel `data` berisi list dengan banyak elemen duplikasi
- Data list kemudian dikonversi ke bentuk set dengan cara membungkus variabelnya menggunakan fungsi `set()`. Operasi ini menghasilkan nilai set berisi elemen unik.
- Selanjutnya data set dikonversi lagi ke bentuk list menggunakan fungsi `list()`.

● Mengecek apakah element ada

Selain untuk kasus di atas, set juga bisa digunakan untuk pengecekan membership dengan kombinasi keyword `if` dan `in`.

Pada contoh berikut, variabel `fellowship` dicek apakah berisi string `gimli` atau tidak.

```
fellowship = {'aragorn', 'gimli', 'legolas'}
to_find = 'gimli'

if to_find in fellowship:
    print(to_find, 'is exists within fellowship')
```

A.15.4. Operasi pada set

● Menambah element

Method `add()` milik tipe data set digunakan untuk menambahkan element baru. O iya, perlu diingat bahwa tipe data ini didesain untuk mengabaikan urutan elemen, jadi urutan tersimpannya elemen bisa saja acak.

```
fellowship = set()

fellowship.add('aragorn')
print("len:", len(fellowship), "data:", fellowship)
# output → len: 1 data: {'aragorn'}

fellowship.add('gimli')
print("len:", len(fellowship), "data:", fellowship)
# output → len: 2 data: {'gimli', 'aragorn'}

fellowship.add('legolas')
print("len:", len(fellowship), "data:", fellowship)
# output → len: 3 data: {'gimli', 'legolas', 'aragorn'}
```

● Menghapus element secara acak

Gunakan method `pop()` untuk menghapus satu elemen secara acak atau random.

```
fellowship = {'narya', 'nenya', 'nilya'}

fellowship.pop()
print("len:", len(fellowship), "data:", fellowship)
# output → len: 2 data: {'narya', 'nilya'}

fellowship.pop()
print("len:", len(fellowship), "data:", fellowship)
# output → len: 1 data: {'nilya'}

fellowship.pop()
print("len:", len(fellowship), "data:", fellowship)
# output → len: 0 data: set()
```

● Menghapus spesifik elemen

Ada dua method tersedia untuk kebutuhan menghapus elemen tertentu dari suatu set, yaitu `discard()` dan `remove()`. Penggunaan keduanya adalah sama, harus disertai dengan 1 argument pemanggilan method, yaitu elemen yang ingin dihapus.

Pada contoh berikut, elemen `boromir` dihapus dari set menggunakan method `discard()`, dan elemen `gandalf` dihapus menggunakan method `remove()`.

```
fellowship = {'aragorn', 'gimli', 'legolas', 'gandalf', 'boromir', 'frodo', 'sam', 'merry', 'pippin'}
print("fellowship:", fellowship)
# output → fellowship: {'aragorn', 'gimli', 'legolas', 'gandalf', 'boromir', 'frodo', 'sam', 'merry', 'pippin'}

fellowship.discard('boromir')
print("fellowship:", fellowship)
# output → fellowship: {'legolas', 'pippin', 'sam', 'aragorn', 'gimli', 'frodo', 'gandalf', 'merry'}

fellowship.remove('gandalf')
print("fellowship:", fellowship)
# output → fellowship: {'legolas', 'pippin', 'sam', 'aragorn', 'gimli', 'frodo', 'merry'}
```

Perbedaan dua method di atas: jika elemen yang ingin dihapus tidak ada, method `discard()` tidak memunculkan error, sedangkan method `remove()` memunculkan error. Contoh:

```
fellowship = {'aragorn', 'gimli', 'legolas', 'gandalf', 'boromir', 'frodo', 'sam', 'merry', 'pippin'}
print("fellowship:", fellowship)

fellowship.discard('batman')
print("fellowship:", fellowship)

fellowship.remove('superman')
print("fellowship:", fellowship)
```

```
-----
KeyError                                Traceback (most recent call last)
d:\Labs\Adam Studio\Ebook\dasarprogramanpython\dasarprogramanpython\examples\sets\main_3.py
  40 fellowship.discard('batman')
  41 print("fellowship:", fellowship)
--> 43 fellowship.remove('superman')
  44 print("fellowship:", fellowship)

KeyError: 'superman'
```

● Mengosongkan isi set

Method `clear()` digunakan untuk mengosongkan isi set.

```
fellowship = {'aragorn', 'gimli', 'legolas'}
fellowship.clear()

print("len:", len(fellowship), "data:", fellowship)
# output → len: 0 data: set()
```

● Copy set

Method `copy()` digunakan untuk meng-copy set, menghasilkan adalah data set baru.

```
data1 = {'aragorn', 'gimli', 'legolas'}
```

Pada contoh di atas, statement `data1.copy()` menghasilkan data baru dengan isi sama seperti isi `data1` ditampung oleh variabel bernama `data2`.

Operasi copy disini jenisnya adalah shallow copy.

Lebih detailnya mengenai shallow copy vs deep copy dibahas pada chapter terpisah.

● Pengecekan *difference* antar set

Method `difference()` digunakan untuk mencari perbedaan elemen antara data (dimana method dipanggil) vs. data pada argument pemanggilan method tersebut.

Sebagai contoh, pada variabel `fellowship` berikut akan dicari elemen yang tidak ada di variabel `hobbits`.

```
fellowship = {'aragorn', 'gimli', 'legolas', 'gandalf', 'boromir', 'frodo', 'sam', 'merry', 'pippin'}
hobbits = {'frodo', 'sam', 'merry', 'pippin', 'bilbo'}

diff = fellowship.difference(hobbits)
print("diff:", diff)
# output → diff: {'boromir', 'legolas', 'aragorn', 'gimli', 'gandalf'}
```

Selain method di atas, adalagi method `difference_update()` yang kegunaannya adalah mengubah nilai data (dimana method dipanggil) dengan nilai baru yang didapat dari perbedaan elemen antara data tersebut vs. data pada argument pemanggilan method.

```
fellowship.difference_update(hobbits)
print("fellowship:", fellowship)
# output → fellowship: {'boromir', 'legolas', 'aragorn', 'gimli', 'gandalf'}
```

● Pengecekan *intersection* antar set

Method `intersection()` digunakan untuk mencari elemen yang ada di data (dimana method dipanggil) vs. data pada argument pemanggilan method tersebut.

Pada variabel `fellowship` berikut akan dicari elemen yang juga ada pada variabel `hobbits`.

```
fellowship = {'aragorn', 'gimli', 'legolas', 'gandalf', 'boromir', 'frodo', 'sam', 'merry', 'pippin'}
hobbits = {'frodo', 'sam', 'merry', 'pippin', 'bilbo'}

diff = fellowship.intersection(hobbits)
print("diff:", diff)
# output → duplicates: {'frodo', 'pippin', 'sam', 'merry'}
```

Tersedia juga method `intersection_update()` yang berguna untuk mengubah nilai data (dimana method dipanggil) dengan nilai baru yang didapat dari kesamaan elemen antara data tersebut vs. data pada argument pemanggilan method.

```
fellowship.intersection_update(hobbits)
print("fellowship:", fellowship)
# output → fellowship: {'frodo', 'pippin', 'sam', 'merry'}
```

● Pengecekan keanggotaan *subset*

Di awal chapter ini kita telah sedikit menyinggung pengecekan membership menggunakan kombinasi keyword `if` dan `in`. Selain metode tersebut, ada alternatif cara lain yang bisa digunakan untuk mengecek apakah suatu data (yang pada konteks ini adalah set) merupakan bagian dari element set lain, caranya menggunakan method `issubset()`.

Method `issubset()` menerima argument berupa data set. Contohnya bisa dilihat pada kode berikut:

```
fellowship = {'aragorn', 'gimli', 'legolas', 'gandalf', 'boromir', 'frodo', 'sam', 'merry', 'pippin'}

hobbits_1 = {'frodo', 'sam', 'merry', 'pippin', 'bilbo'}
res_1 = hobbits_1.issubset(fellowship)
print("res_1:", res_1)
# output → res_1: False

hobbits_2 = {'frodo', 'sam', 'merry', 'pippin'}
res_2 = hobbits_2.issubset(fellowship)
print("res_2:", res_2)
# output → res_2: True
```

- Nilai `res_1` adalah `False` karena set `hobbits_1` memiliki setidaknya satu elemen yang bukan anggota dari `fellowship`, yaitu `bilbo`.
- Nilai `res_2` adalah `True` karena set `hobbits_2` semua elemennya adalah anggota dari `fellowship`.

● Pengecekan keanggotaan *superset*

Selain `issubset()`, ada juga `issuperset()` yang fungsinya kurang lebih sama namun kondisinya pengecekannya dibalik.

Agar lebih jelas, silakan lihat kode berikut:

```
fellowship = {'aragorn', 'gimli', 'legolas', 'gandalf', 'boromir', 'frodo', 'sam', 'merry', 'pippin'}

hobbits_1 = {'frodo', 'sam', 'merry', 'pippin', 'bilbo'}
res_1 = fellowship.issuperset(hobbits_1)
print("res_1:", res_1)
# output → res_1: False

hobbits_2 = {'frodo', 'sam', 'merry', 'pippin'}
res_2 = fellowship.issuperset(hobbits_2)
print("res_2:", res_2)
# output → res_2: True
```

- Nilai `res_1` adalah `False` karena set `hobbits_1` memiliki setidaknya satu elemen yang bukan anggota dari

`fellowship`, yaitu `bilbo`.

- Nilai `res_2` adalah `True` karena set `hobbits_2` semua elemennya adalah anggota dari `fellowship`.

● Pengecekan keanggotaan *disjoint*

Method ini mengembalikan nilai `True` jika set pada pemanggilan fungsi berisi elemen yang semuanya bukan anggota data dimana method dipanggil.

```
fellowship = {'aragorn', 'gimli', 'legolas', 'gandalf', 'boromir', 'frodo', 'sam', 'merry', 'pippin'}

res_1 = fellowship.isdisjoint({'aragorn', 'gimli'})
print("res_1:", res_1)

res_2 = fellowship.isdisjoint({'pippin', 'bilbo'})
print("res_2:", res_2)

res_3 = fellowship.isdisjoint({'bilbo'})
print("res_3:", res_3)
```

- Nilai `res_1` adalah `False` karena beberapa anggota set `fellowship` adalah `aragorn` dan `gimli`.
- Nilai `res_2` adalah `False` karena beberapa anggota set `fellowship` adalah `pippin`. Sedangkan `bilbo` ia bukanlah anggota `fellowship`, tapi karena setidaknya ada 1 elemen yang match, maka method `isdisjoint` mengembalikan nilai `False`.
- Nilai `res_3` adalah `True` karena `bilbo` bukanlah anggota `fellowship`.

● *Extend/concat/union* element

Operasi *extend* (atau *concat* atau *union*) adalah operasi penggabungan dua data set. Ada beberapa metode yang tersedia, diantaranya:

- via method `union()`:

```
hobbits = {'frodo', 'sam', 'merry', 'pippin'}
dunedain = {'aragorn'}
elf = {'legolas'}
dwarf = {'gimli'}
human = {'boromir'}
maiar = {'gandalf'}

fellowship_1 = hobbits.union(dunedain).union(dunedain).union(elf).union(dwarf).union(human).union(maiar)
print("fellowship_1:", fellowship_1)
# output → fellowship_1: {'boromir', 'gimli', 'legolas', 'pippin', 'sam', 'aragorn', 'frodo', 'gandalf', 'merry'}
```

- via method `update()`:

```
hobbits = {'frodo', 'sam', 'merry', 'pippin'}
dunedain = {'aragorn'}
elf = {'legolas'}
```


Bisa dilihat perbedaannya ada di-bagaimana nilai balik method disimpan.

- Pada method `union()`, pemanggilan method tersebut mengembalikan data setelah penggabungan, dan bisa di-chain langsung dengan pemanggilan method `union()` lainnya.
- Pada method `update()`, data yang digunakan untuk memanggil method tersebut diubah secara langsung nilainya.

● Operator bitwise pada set

- Operasi `or` pada set menggunakan operator `|`

```
a = set('abracadabra') # {'c', 'a', 'r', 'd', 'b'}
b = set('alacazam')    # {'c', 'z', 'a', 'm', 'l'}

res = a | b
print(res) # output → {'c', 'z', 'a', 'r', 'd', 'b', 'm', 'l'}
```

Nilai `res` berisi elemen set unik kombinasi set `a` dan set `b`.

- Operasi `and` pada set menggunakan operator `&`

```
a = set('abracadabra') # {'c', 'a', 'r', 'd', 'b'}
b = set('alacazam')    # {'c', 'z', 'a', 'm', 'l'}

res = a & b
print(res) # output → {'c', 'a'}
```

Nilai `res` berisi elemen set yang merupakan anggota set `a` dan set `b`. Operasi seperti ini biasa disebut dengan operasi *and*.

- Operasi `exclusive or` pada set menggunakan operator `^`

```
a = set('abracadabra') # {'c', 'a', 'r', 'd', 'b'}
b = set('alacazam')    # {'c', 'z', 'a', 'm', 'l'}

res = a ^ b
print(res) # output → {'z', 'r', 'b', 'd', 'm', 'l'}
```

Nilai `res` berisi elemen set yang ada di set `a` atau set `b` tetapi tidak ada di-keduanya.

● Operator `-` pada set

Digunakan untuk pencarian perbedaan elemen. Contoh penerapan:

```
a = set('abracadabra') # {'c', 'a', 'r', 'd', 'b'}
b = set('alacazam')    # {'c', 'z', 'a', 'm', 'l'}

res = a - b
```

Nilai `res` berisi elemen set unik yang merupakan anggota set `a` tapi bukan anggota set `b`

A.15.5. Fungsi `set()`

● Konversi string ke set

String dibungkus menggunakan method `set()` menghasilkan data sets berisi karakter string yang unik.

```
data = set('abcda')
print('data', data)
# output → data {'c', 'b', 'a', 'd'}
```

● Konversi list ke set

Data list bisa diubah menjadi set dengan mudah dengan cara membungkusnya menggunakan fungsi `set()`. Isi dari set adalah elemen unik list.

```
data = set(['a', 'b', 'c', 'd', 'a'])
print('data', data)
# output → data {'c', 'b', 'a', 'd'}
```

● Konversi tuple ke set

Data tuple juga bisa diubah menjadi set via fungsi `set()`. Isi dari set adalah elemen unik tuple.

```
data = set(('a', 'b', 'c', 'd', 'a'))
print('data', data)
# output → data {'c', 'b', 'a', 'd'}
```

● Konversi range ke set

Data range (hasil dari pemanggilan fungsi `range()`) bisa dikonversi ke bentuk set via fungsi `set()`.

```
data = set(range(1, 5))
print('data', data)
# output → data {1, 2, 3, 4}
```

A.15.6. Set comprehension

Metode `comprehension` juga bisa diterapkan pada set. Contohnya bisa dilihat pada kode berikut, statement set comprehension dibuat untuk melakukan pengecekan apakah ada element pada set `set('abracadabra')` yang bukan anggota element `set('abc')`.

```
res = {x for x in set('abracadabra') if x not in set('abc')}  
print(res)  
# output → {'d', 'r'}
```

A.15.7. frozenset

`frozenset` adalah `set` yang *immutable* atau tidak bisa diubah nilai elemennya setelah dideklarasikan.

Cara penggunaannya seperti `set`, perbedaannya pada deklarasi `frozenset`, fungsi `frozenset()` digunakan dan bukan `set()`.

```
a = frozenset('abracadabra')  
print(a)  
# output → frozenset({'c', 'a', 'r', 'd', 'b'})  
  
b = frozenset('alacazam')  
print(b)  
# output → frozenset({'c', 'z', 'a', 'm', 'l'})
```

Semua operasi `set`, method milik `set` bisa digunakan pada `frozenset`, kecuali beberapa operasi yang sifatnya *mutable* atau mengubah elemen. Contohnya seperti method `add()`, `pop()`, `remove()` dan lainnya tidak bisa digunakan di `frozenset`.

Catatan chapter

● Source code praktik

github.com/nova1agung/dasarpemrogramanpython-example/blob/master/set

● Chapter relevan lainnya

- [List](#)
- [List Comprehension](#)
- [Tuple](#)

● Referensi

- <https://docs.python.org/3/tutorial/datastructures.html>
 - <https://docs.python.org/3/library/stdtypes.html#set-types-set-frozenset>
-

A.16. Python Dictionary

Pada chapter ini kita akan belajar salah satu tipe data *mapping* di Python, yaitu Dictionary.

A.16.1. Penerapan Dictionary

Dictionary atau `dict` adalah tipe data kolektif berbentuk **key-value**. Contoh penulisannya:

```
profile = {  
    "id": 2,  
    "name": "john wick",  
    "hobbies": ["playing with pencil"],  
    "is_female": False,  
}
```

Literal dictionary ditulis dengan menggunakan `{ }`, mirip seperti `Set`, hanya saja bedanya pada tipe dictionary isinya berbentuk **key-value**.

Ok, sekarang dari kode di atas, coba tambahkan kode berikut untuk melihat bagaimana data dictionary dimunculkan di layar console.

```
print("data:", profile)  
print("total keys:", len(profile))
```

```
{'id': 2, 'name': 'john wick', 'hobbies': ['playing with pencil'], 'is_female': False}  
total keys: 4
```

Sedangkan untuk memunculkan nilai item tertentu berdasarkan key-nya, bisa

dilakukan menggunakan notasi `dict["key"]`. Contoh:

```
print("name:", profile["name"])  
# output → name: john wick  
  
print("hobbies:", profile["hobbies"])  
# output → ['playing with pencil']
```

DANGER

Pengaksesan item menggunakan key yang tidak dikenali akan menghasilkan error.

Sebagai contoh, variabel `profile` di atas jika diakses item dengan key `umur` misalnya (`profile["umur"]`) hasilnya adalah error.

● Urutan item dictionary

Mulai dari Python version 3.7, item dictionary tersimpan secaraurut. Artinya urutan item dictionary akan selalu sesuai dengan bagaimana inisialisasi awalnya.

● Pretty print dictionary

Ada tips agar data dictionary yang di-print di console muncul dengan tampilan yang lebih mudah dibaca, dua diantaranya:

- Menggunakan `pprint.pprint()`:

Import terlebih dahulu module `pprint`, lalu gunakan fungsi `pprint()` untuk memunculkan data ke console.

```
import pprint
pprint.pprint(profile)
```

```
{'hobbies': ['playing with pencil'],
 'id': 2,
 'is_female': False,
 'name': 'john wick'}
```

- Menggunakan `json.dumps()`:

Import terlebih dahulu module `json`, lalu gunakan fungsi `dumps()` untuk memformat dictionary menjadi bentuk string yang mudah dibaca, kemudian print menggunakan `print()`.

Tentukan lebar *space indentation* sesuai selera (pada contoh di bawah ini di set nilainya `4` spasi).

```
import json
print(json.dumps(profile, indent=4))
```

```
{
    "id": 2,
    "name": "john wick",
    "hobbies": [
        "playing with pencil"
    ],
    "is_female": false
}
```

A.16.2. Inisialisasi dictionary

Pembuatan data dictionary bisa dilakukan menggunakan beberapa cara:

- Menggunakan `{ }`:

```
profile = {  
    "id": 2,  
    "name": "john wick",  
    "hobbies": ["playing with pencil"],  
    "is_female": False,  
}
```

- Menggunakan fungsi `dict()` dengan isi argument **key-value**:

```
profile = dict(  
    set="id",  
    name="john wick",  
    hobbies=["playing with pencil"],  
    is_female=False,  
)
```

- Menggunakan fungsi `dict()` dengan isi list tuple:

```
profile = dict([  
    ('set', "id"),  
    ('name', "john wick"),  
    ('hobbies', ["playing with pencil"]),  
    ('is_female', False)  
)
```

Sedangkan untuk membuat dictionary tanpa item atau kosong, bisa cukup menggunakan `dict()` atau `{ }`:

```
profile = dict()  
print(profile)
```

A.16.3. Perulangan item dictionary

Gunakan keyword `for` dan `in` untuk mengiterasi data tiap key milik dictionary. Dari key tersebut kemudian akses value-nya.

```
profile = {  
    "id": 2,  
    "name": "mario",  
    "hobbies": ("playing with luigi", "saving the mushroom kingdom"),  
    "is_female": False,  
}  
  
for key in profile:  
    print("key:", key, "\t value:", profile[key])
```

Karakter `\t` menghasilkan tab. Penggunaan karakter ini bisa membuat rapi tampilan output.

Program di atas ketika di run outputnya:

```
key: id          value: 2  
key: name        value: mario  
key: hobbies     value: ('playing with luigi', 'saving the mushroom kingdom')  
key: is_female   value: False
```

A.15.4. Nested dictionary

Dictionary bercabang atau **nested dictionary** bisa dimanfaatkan untuk menyimpan data dengan struktur yang kompleks, misalnya dictionary yang salah satu value item-nya adalah list.

Penerapannya tak berbeda seperti inisialisasi dictionary umumnya, langsung tulis saja dictionary sebagai child dictionary. Contoh:

```
profile = {
    "id": 2,
    "name": "mario",
    "hobbies": ("playing with luigi", "saving the mushroom kingdom"),
    "is_female": False,
    "affiliations": [
        {
            "name": "luigi",
            "affiliation": "brother"
        },
        {
            "name": "mushroom kingdom",
            "affiliation": "protector"
        },
    ]
}

print("name:", profile["name"])
print("hobbies:", profile["hobbies"])
print("affiliations:")

for item in profile["affiliations"]:
    print("    → %s (%s)" % (item["name"], item["affiliation"]))

# output ↓
#
# name: mario
# hobbies: ('playing with luigi', 'saving the mushroom kingdom')
# affiliations:
#    → luigi (brother)
#    → mushroom kingdom (protector)
```

Pada kode di atas, key `affiliations` berisi array object dictionary.

Contoh cara mengakses value nested item dictionary:

```
value = profile["affiliations"][0]["name"],
profile["affiliations"][0]["affiliation"]
print(" → %s (%s)" % (value))
# output → luigi (brother)

value = profile["affiliations"][1]["name"],
profile["affiliations"][1]["affiliation"]
print(" → %s (%s)" % (value))
# output → mushroom kingdom (protector)
```

A.15.5. Dictionary mutability

Item dictionary adalah mutable, perubahan value item bisa dilakukan langsung menggunakan operator assignment `=`.

```
profile = {
    "id": 2,
    "name": "mario",
    "hobbies": ("playing with luigi", "saving the mushroom kingdom"),
    "is_female": False
}

print(profile["affiliations"][0]["name"])
# output → luigi

profile["affiliations"][0]["name"] = "luigi steven"

print(profile["affiliations"][0]["name"])
# output → luigi steven
```

A.15.6. Operasi data dictionary

● Pengaksesan item

Pengaksesan item dilakukan lewat notasi `dict["key"]`, atau bisa dengan menggunakan method `get()`.

```
profile = {
    "id": 2,
    "name": "mario",
    "hobbies": ("playing with luigi", "saving the mushroom kingdom"),
    "is_female": False,
}

print("id:", profile["id"])
# output → id: 2

print("name:", profile.get("name"))
# output → name: mario
```

● Mengubah isi dictionary

Cara mengubah value item dictionary adalah dengan mengaksesnya terlebih dahulu, kemudian diikuti operasi assignment.

```
profile = {
    "id": 2,
    "name": "mario",
    "hobbies": ("playing with luigi", "saving the mushroom kingdom"),
    "is_female": False,
}

print("name:", profile["name"])
```

● Menambah item dictionary

Caranya adalah mirip seperti operasi pengubahan value item, perbedaannya ada pada key-nya. Key yang ditulis adalah key item baru yang.

```
profile = {  
    "name": "mario",  
}  
print("len:", len(profile), "data:", profile)  
# output → len: 1 data: {'name': 'mario'}  
  
profile["favourite_color"] = "red"  
print("len:", len(profile), "data:", profile)  
# output → len: 2 data: {'name': 'mario', 'favourite_color': 'red'}
```

Selain cara tersebut, bisa juga dengan menggunakan method `update()`. Tulis key dan value baru yang ingin ditambahkan sebagai argument method `update()` dalam bentuk dictionary.

```
profile.update({"race": "italian"})  
print("len:", len(profile), "data:", profile)  
# output → len: 3 data: {'name': 'mario', 'favourite_color': 'red',  
'race': 'italian'}
```

● Menghapus item dictionary

Method `pop()` digunakan untuk menghapus item dictionary berdasarkan key.

```
profile.pop("hobbies")  
print(profile)
```

Keyword `del` juga bisa difungsikan untuk operasi yang sama. Contoh:

```
del profile["id"]
print(profile)
```

● Pengaksesan dictionary keys

Method `keys()` digunakan untuk mengakses semua keys dictionary, hasilnya adalah tipe data view objects `dict_keys`. Dari nilai tersebut bungkus menggunakan `list()` untuk mendapatkan nilainya dalam bentuk list.

```
profile = {
    "id": 2,
    "name": "mario",
    "hobbies": ("playing with luigi", "saving the mushroom kingdom"),
    "is_female": False,
}

print(list(profile.keys()))
# output → ['id', 'name', 'is_female']
```

● Pengaksesan dictionary

Method `values()` digunakan untuk mengakses semua keys dictionary, hasilnya adalah tipe data view objects `dict_values`. Gunakan fungsi `list()` untuk mengkonversinya ke bentuk list.

```
profile = {
    "id": 2,
    "name": "mario",
    "hobbies": ("playing with luigi", "saving the mushroom kingdom"),
    "is_female": False,
```

● Method `items()` dictionary

Digunakan untuk mengakses semua keys dictionary. Nilai baliknya bertipe view objects `dict_items` yang strukturnya cukup mirip seperti list berisi tuple.

Untuk mengkonversinya ke bentuk list, gunakan fungsi `list()`.

```
profile = {
    "id": 2,
    "name": "mario",
    "hobbies": ("playing with luigi", "saving the mushroom kingdom"),
    "is_female": False,
}

print(list(profile.items()))
# output → [('id', 2), ('name', 'mario'), ('is_female', False)]
```

● Copy dictionary

Method `copy()` digunakan untuk meng-copy dictionary, hasilnya data dictionary baru.

```
p1 = {
    "id": 2,
    "name": "mario",
    "is_female": False,
}
print(p1)
# output → {'id': 2, 'name': 'mario', 'is_female': False}

p2 = p1.copy()
print(p2)
# output → {'id': 2, 'name': 'mario', 'is_female': False}
```

Pada contoh di atas, statement `p1.copy()` menghasilkan data baru dengan isi sama seperti isi `p1`, data tersebut kemudian ditampung oleh variabel `p2`.

Operasi copy disini jenisnya adalah shallow copy.

Lebih detailnya mengenai shallow copy vs deep copy dibahas pada chapter terpisah.

● Mengosongkan isi dictionary

Method `clear()` berguna untuk menghapus isi dictionary.

```
profile = {
    "id": 2,
    "name": "mario",
    "is_female": False,
}
print("len:", len(profile), "data:", profile)
# output → len: 3 data: {'id': 2, 'name': 'mario', 'is_female': False}

profile.clear()
print("len:", len(profile), "data:", profile)
# output → len: 0 data: {}
```

Catatan chapter

● Source code praktik

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

● Referensi

- <https://docs.python.org/3/library/stdtypes.html#mapping-types-dict>
-

A.17. Python String

String (atau `str`) merupakan kumpulan data `char` atau karakter yang tersimpan secara urut (*text sequence*). String di Python mengadopsi standar Unicode dengan *default encoding* adalah `UTF-8`.

A.17.1. Penerapan string

Python mendesain tipe data string dalam bentuk yang sangat sederhana dan mudah digunakan. Untuk membuat string cukup tulis saja text yang diinginkan dengan diapit tanda petik satu atau petik dua. Contoh:

```
str = "hello python"
print(str)
# output → hello python

str = 'hello python'
print(str)
# output → hello python
```

● Multiline string

Untuk string *multiline* atau lebih dari satu baris, cara penulisannya bisa dengan:

- Menggunakan karakter spesial `\n`:

```
str = "a multiline string\nin python"

print(str)
```

- Atau menggunakan tanda `"""` untuk mengapit text. Contoh:

```
str = """a multiline string
in python"""

print(str)
# output ↓
# a multiline string
# in python
```

● Escape character

Python mengenal *escape character* umum yang ada di banyak bahasa pemrograman, contohnya seperti `\` digunakan untuk menuliskan karakter `"` (pada string yang dibuat menggunakan literal `" "`). Penambahan karakter `\` adalah penting agar karakter `"` terdeteksi sebagai penanda string.

Sebagai contoh, dua statement berikut adalah ekuivalen:

```
str = 'this is a "string" in python'
print(str)
# output → this is a "string" in python

str = "this is a \"string\" in python"
print(str)
# output → this is a "string" in python
```

A.17.2. String *special characters*

Di atas telah dicontohkan bagaimana cara menulis karakter *newline* atau baris baru menggunakan `\n`, dan karakter petik dua menggunakan `\"`. Dua

karakter tersebut adalah contoh dari *special characters*.

Python mengenal banyak special characters yang masing-masing memiliki kegunaan yang cukup spesifik. Agar lebih jelas silakan lihat tabel berikut:

Special character	Kegunaan
<code>\\</code>	karakter backslash (<code>\</code>)
<code>\'</code>	tanda petik satu (<code>'</code>)
<code>\"</code>	tanda kutip (petik dua) (<code>"</code>)
<code>\a</code>	bunyi <i>beeb</i> (ASCII BEL)
<code>\b</code>	backspace (ASCII BS)
<code>\f</code>	page separator / formfeed (ASCII FF)
<code>\n</code>	karakter baris baru linefeed (ASCII LF)
<code>\r</code>	karakter baris baru carriage return (ASCII CR)
<code>\t</code>	horizontal tab (ASCII TAB)
<code>\v</code>	vertical tab (ASCII VT)
<code>\{oktal}</code>	nilai oktal, contoh: <code>\122</code> , <code>\004</code> , <code>\024</code>
<code>\x{hex}</code>	nilai heksadesimal, contoh: <code>\xA4</code> , <code>\x5B</code>

Tambahan contoh penggunaan salah satu special character `\t` (horizontal tab):

```
print("Nama\t\t| Umur\t| Gender")
print("-----")
print("Bruce Wayne\t| 34\t| laki-laki")
print("Cassandra Cain\t| 22\t| perempuan")
```

Program di atas menghasilkan output berikut:

```
..  Nama          | Umur  | Gender
   -----
   Bruce Wayne   | 34    | laki-laki
   Cassandra Cain| 22    | perempuan
```

Syntax `0xC548` adalah salah satu penulisan numerik berbasis hexadecimal. Lebih jelasnya dibahas pada chapter *Numeric*.

A.17.3. String formatting

String formatting adalah teknik untuk mem-format string agar menghasilkan text sesuai dengan format yang diinginkan.

Cara termudah melakukan string formatting adalah dengan menggunakan **f-strings** (atau *formatted string literals*). Tulis string seperti biasa tapi diawali dengan huruf `f` atau `F` sebelum penulisan `" "`.

Pada contoh berikut, sebuah string dibuat dimana dua bagian string didalamnya datanya bersumber dari variabel string lain.

```
name = "Aiden Pearce"
occupation = "IT support"

str = f"hello, my name is {name}, I'm an {occupation}"
print(str)
# output → hello, my name is Aiden Pearce, I'm an IT support
```

Penjelasan:

- String dibuat dengan metode f-strings, dimana struktur text adalah `hello, my name is {name}, I'm an {occupation}`.
- Text `{name}` di dalam string di-replace oleh nilai variable `name`, yang pada konteks ini nilainya `Aiden Pearce`.
- Text `{occupation}` di dalam string di-replace oleh nilai variable `occupation`, yang pada konteks ini nilainya `IT support`.
- f-strings di atas menghasilkan text `hello, my name is Aiden Pearce, I'm an IT support`.

*Pada penerapan metode **f-strings**, isi dari `{}` tidak harus data string, tetapi tipe data lainnya juga bisa digunakan salahkan printable atau bisa di-print.*

Selain menggunakan metode di atas, ada beberapa alternatif cara lain yang bisa digunakan, diantaranya:

```
str = "hello, my name is {name}, I'm an {occupation}".format(name = name,
occupation = occupation)
print(str)
# output → hello, my name is Aiden Pearce, I'm an IT support

str = "hello, my name is {0}, I'm an {1}".format(name, occupation)
```

Semua metode string formatting yang telah dipelajari menghasilkan nilai balik yang sama, yaitu `hello, my name is Aiden Pearce, I'm an IT support`. Mana yang lebih baik? Silakan pilih saja metode yang sesuai selera.

A.17.4. Penggabungan string (*concatenation*)

Ada beberapa metode yang bisa digunakan untuk *string concatenation* atau operasi penggabungan string.

- Menggunakan teknik penulisan string literal sebaris.

Caranya dengan langsung tulis saja semua string-nya menggunakan separator karakter spasi.

```
str = "hello " "python"
print(str)
# output → hello python
```

- Menggunakan operator `+`.

Operator `+` jika diterapkan pada string menghasilkan penggabungan string.

```
str_one = "hello"
str_two = "python"
str = str_one + " " + str_two

print(str)
# output → hello python
```

- Menggunakan method `join()` milik string.

Pada penerapannya, karakter pembatas atau *separator* ditulis terlebih dahulu, kemudian di-*chain* dengan method join dengan isi argument adalah list yang ingin digabung.

```
str = " ".join(["hello", "python"])
print(str)
# output → hello python
` ``
```

A.17.5. Operasi sequence pada string

String masih termasuk kategori tipe data sequence, yang artinya bisa digunakan pada operasi standar sequence, contoh seperti perulangan, pengaksesan elemen, dan slicing.

● Mengecek lebar karakter string

Fungsi `len()` ketika digunakan pada tipe data string mengembalikan informasi jumlah karakter string.

```
str = "hello python"

print("text:", str)
# output → hello python

print("length:", len(str))
# output → 12
```

● Mengakses element string

Setiap elemen string bisa diakses menggunakan index. Penulisan notasi

pengaksesannya sama seperti pada tipe data sequence lainnya, yaitu menggunakan `str[index]`.

```
str = "hello python"
print(str[0])
# output → h

print(str[1])
# output → e

print(str[2])
# output → l
```

Selain via index, keyword perulangan `for` bisa dimanfaatkan untuk mengiterasi elemen string. Contoh:

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

Contoh lain menggunakan `range()`:

```
for i in range(0, len(str)):
    print(str[i])
```

Output:


```
h  
e  
l  
l  
o  
  
p  
y  
t  
h  
o  
n
```

DANGER

Pengaksesan elemen menggunakan index di-luar kapasitas data akan menghasilkan error.

Sebagai contoh, string `str = "hello"`, jika diakses index ke-5-nya misalnya (`str[5]`) hasilnya adalah error.

● ***Slicing string***

Teknik slicing bisa diterapkan pada data string. Contoh:

```
str = "hello python"
```

```
print(str[1:5])
```

```
# output → ello
```

```
print(str[7:])
```

```
# output → ython
```

```
print(str[:4])
```

```
# output → hell
```

Lebih detailnya mengenai slice dibahas pada chapter *Slice*

A.17.6. Operasi *character* & *case*

Tipe data `str` memiliki beberapa method yang berguna untuk keperluan operasi string yang berhubungan dengan *character* & *case*

● Pengecekan karakter alfabet dan angka

- Method `isalpha()` digunakan untuk mengecek apakah string berisi karakter alfabet atau tidak. Nilai kembaliannya `True` jika semua karakter dalam string adalah alfabet.

```
print("abcdef".isalpha())
# output → True, karena abcdef adalah alfabet

print("abc123".isalpha())
# output → False, karena ada karakter 123 yang bukan merupakan alfabet

print("موز".isalpha())
# output → True, karena موز adalah abjad arabic

print("◇◇◇".isalpha())
# output → True, karena ◇◇◇ adalah karakter jepang
```

- Method `isdigit()` digunakan untuk mengecek apakah string berisi karakter digit atau tidak. Nilai kembaliannya `True` jika semua karakter dalam string adalah angka numerik (termasuk pangkat).

```
print("123456".isdigit())
# output → True, karena 123456 adalah digit
```

- Method `isdecimal()` digunakan untuk mengecek apakah string berisi karakter desimal atau tidak. Nilai kembaliannya `True` jika semua karakter dalam string adalah angka numerik desimal.

```
print("123456".isdecimal())
# output → True, karena 123456 adalah angka desimal

print("123abc".isdecimal())
# output → False, karena ada karakter abc yang bukan merupakan angka desimal

print('2½'.isdecimal())
# output → False, karena bilangan pecahan memiliki karakter `/'` yang tidak termasuk dalam kategori angka desimal

print('4²'.isdecimal())
# output → False, karena bilangan pangkat yang tidak termasuk dalam kategori angka desimal

print('٢٨'.isdecimal())
# output → True, karena ٢٨ adalah angka desimal arabic

print('4'.isdecimal())
# output → True, karena 4 adalah angka desimal
```

- Method `isnumeric()` digunakan untuk mengecek apakah string berisi karakter desimal atau tidak. Nilai kembaliannya `True` jika semua karakter dalam string adalah angka numerik (termasuk pecahan, pangkat, dan angka numerik lainnya).

```
print("123456".isnumeric())
# output → True, karena 123456 adalah angka numerik

print("123abc".isnumeric())
```

- Method `isalnum()` digunakan untuk mengecek apakah string berisi setidaknya karakter alfabet atau digit, atau tidak keduanya. Nilai kembaliannya `True` jika semua karakter dalam string adalah alfabet atau angka numerik.

```
print("123abc".isalnum())
# output → True, karena 123 adalah digit dan abc adalah alfabet

print("12345½".isalnum())
# output → True, karena 12345½ adalah digit

print("abcdef".isalnum())
# output → True, karena abcdef adalah alfabet

print("abc 12".isalnum())
# output → False, karena ada karakter spasi yang bukan merupakan
karakter digit ataupun alfabet

print("موز".isalnum())
# output → True, karena موز adalah abjad arabic

print("💎💎💎".isalnum())
# output → True, karena 💎💎💎 adalah karakter jepang
```

● Pengecekan karakter *whitespace*

Method `isspace()` digunakan untuk mengecek apakah string berisi karakter *whitespace*.

```
print(" ".isspace())
# output → True, karena string berisi karakter spasi

print("\n".isspace())
# output → True, karena string berisi karakter newline
```

● Pengecekan karakter case

- Method `islower()` digunakan untuk mengecek apakah semua karakter string adalah ditulis dalam huruf kecil (*lower case*), jika kondisi tersebut terpenuhi maka nilai kembaliannya adalah `True`.

```
print("hello python".islower())  
# output → True  
  
print("Hello Python".islower())  
# output → False  
  
print("HELLO PYTHON".islower())  
# output → False
```

- Method `istitle()` digunakan untuk mengecek apakah kata dalam string adalah ditulis dengan awalan huruf besar (*title case*), jika kondisi tersebut terpenuhi maka nilai kembaliannya adalah `True`.

```
print("hello python".istitle())  
# output → False  
  
print("Hello Python".istitle())  
# output → True  
  
print("HELLO PYTHON".istitle())  
# output → False
```

- Method `isupper()` digunakan untuk mengecek apakah semua karakter string adalah ditulis dalam huruf besar (*upper case*), jika kondisi tersebut terpenuhi maka nilai kembaliannya adalah `True`.

```
print("hello python".isupper())  
# output → False  
  
print("Hello Python".isupper())  
# output → False  
  
print("HELLO PYTHON".isupper())  
# output → True
```

● Mengubah karakter *case*

Beberapa method yang bisa digunakan untuk mengubah *case* suatu string:

- Method `capitalize()` berfungsi untuk mengubah penulisan karakter pertama string menjadi huruf besar (*capitalize*).
- Method `title()` berfungsi untuk mengubah penulisan kata dalam string diawali dengan huruf besar (*title case*).
- Method `upper()` berfungsi untuk mengubah penulisan semua karakter string menjadi huruf besar (*upper case*).
- Method `lower()` berfungsi untuk mengubah penulisan semua karakter string menjadi huruf kecil (*lower case*).
- Method `swapcase()` berfungsi untuk membalik penulisan *case* karakter string. Untuk karakter yang awalnya huruf kecil menjadi huruf besar, dan sebaliknya.

```
print("hello python".capitalize())  
# output → Hello python  
  
print("hello python".title())  
# output → Hello Python  
  
print("hello python".upper())
```

A.17.7. Operasi pencarian string & substring

● Pengecekan string menggunakan keyword `in`

Keyword `in` bisa digunakan untuk mengecek apakah suatu string merupakan bagian dari string lain. Nilai balik statement adalah boolean. Contoh:

```
str = "hello world"
print("ello" in str)
# output → True
```

Teknik tersebut bisa dikombinasikan dengan seleksi kondisi `if`:

```
str = "hello world"
if "ello" in str:
    print(f"py is in {str}")
# output → py is in hello world
```

● Pengecekan substring

Ada beberapa Method yang bisa digunakan untuk keperluan pengecekan substring, apakah suatu string merupakan dari string lain.

- Menggunakan method `startswith()` untuk mengecek apakah suatu string diawali dengan huruf/kata tertentu.

```
print("hello world".startswith("hell"))
# output → True
```

- Menggunakan method `endswith()` untuk mengecek apakah suatu string diakhiri dengan huruf/kata tertentu.

```
print("hello world".endswith("world"))  
# output → True  
  
print("hello world".endswith("worl"))  
# output → False
```

- Menggunakan method `count()` untuk mengecek apakah suatu string merupakan bagian dari string lain.

```
print("hello world".count("ello"))  
# output → 1
```

Method ini mengembalikan jumlah huruf/kata yang ditemukan. Jika kebutuhannya adalah mencari tau apakah suatu substring ada atau tidak, maka gunakan operasi logika lebih dari 0 (atau `n > 0`).

```
print("hello world".count("ello") > 0)  
# output → True
```

● Pencarian index substring

Method-method berikut sebenarnya kegunaannya mirip seperti method untuk pengecekan substring, perbedaannya adalah nilai balik pemanggilan method berupa index substring.

- Method `count()` mengembalikan jumlah substring yang ditemukan sesuai kata kunci yang dicari.


```
str = "hello world hello world"
print(str.count("ello"))
# output → 2
```

- Method `index()` mengembalikan index substring pertama yang ditemukan sesuai kata kunci yang dicari. Jika substring tidak ditemukan, method ini menghasilkan error.

```
str = "hello world hello world"
print(str.index("worl"))
# output → 6
```

- Method `rindex()` mengembalikan index substring pertama yang ditemukan sesuai kata kunci yang dicari dengan urutan pencarian adalah dari kanan. Jika substring tidak ditemukan, method ini menghasilkan error.

```
str = "hello world hello world"
print(str.rindex("worl"))
# output → 18
```

- Method `find()` mengembalikan index substring pertama yang ditemukan sesuai kata kunci yang dicari. Jika substring tidak ditemukan, method ini menghasilkan nilai `-1`.

```
str = "hello world hello world"
print(str.find("worl"))
# output → 6
```

- Method `rfind()` mengembalikan index substring pertama yang ditemukan sesuai kata kunci yang dicari dengan urutan pencarian adalah dari kanan. Jika substring tidak ditemukan, method ini menghasilkan nilai

-1.

```
str = "hello world hello world"
print(str.rfind("worl"))
# output → 18
```

A.17.8. Operasi string lainnya

● Replace substring

Method `replace()` digunakan untuk me-replace suatu substring dengan string lain. Contoh penggunaan:

```
str_old = "hello world"
str_new = str_old.replace("world", "python")
print(str_new)
# output → hello python
```

● Trim / strip

Metode trimming/stripping digunakan untuk menghapus *whitespace* yang diantaranya adalah baris baru dan juga spasi.

Sebelum kita mulai, coba perhatikan kode berikut. String `str` dideklarasikan menggunakan `""" """` yang dalam penerapannya tidak akan meng-escape whitespace.

```
str = """
hello python
"""
```

Bisa dilihat saat di print kelihatan *newline* atau baris barunya pada awal string dan juga akhir string.

Dengan menggunakan teknik trimming, *whitespace* bisa dihilangkan. Ada beberapa method yang bisa digunakan, diantaranya:

- Method `lstrip()` untuk trim *whitespace* karakter di awal atau sebelah kiri string.

```
str = """
hello python
"""

print(f"--{str.lstrip()}--")
# output ↓
# --hello python
# --
```

- Method `rstrip()` untuk trim *whitespace* karakter di akhir atau sebelah kanan string.

```
str = """
hello python
"""

print(f"--{str.rstrip()}--")
# output ↓
# --
# hello python--
```

- Method `strip()` untuk trim *whitespace* karakter di awal dan akhir string.

```
str = """
```

● Join string

Method `join()` berguna untuk menggabungkan list berisi element string. String yang digunakan untuk memanggil method ini menjadi *separator* operasi join.

```
data = ["hello", "world", "abcdef"]
res = "-".join(data)
print(res)
# output → hello-world-abcdef
```

Catatan chapter

● Source code praktik

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

● TBA

- Bytes

● Referensi

- <https://docs.python.org/3/library/string.html>
- <https://docs.python.org/3/library/stdtypes.html#text-sequence-type-str>
- https://docs.python.org/3/reference/lexical_analysis.html#f-strings

A.18. Python Unicode

Python mengadopsi aturan standar **Unicode** dalam pengolahan karakter dalam string. Benefitnya Python mendukung dan mengenali berbagai macam jenis karakter, termasuk diantaranya adalah huruf Arab, Jepang, emoji, symbol, dan banyak jenis karakter lainnya.

Unicode sendiri adalah suatu aturan standar untuk encoding text yang di-maintain oleh Unicode Consortium. Standarisasi ini diciptakan untuk mendukung semua jenis penulisan yang ada di bumi.

Pada chapter ini kita akan membahas tentang bagaimana implementasi Unicode di Python.

A.18.1. Penerapan **Unicode**

Dalam dunia per-Unicode-an, ada yang disebut dengan **code point** yaitu suatu angka numerik (bisa desimal maupun hexadecimal) yang merepresentasikan karakter tertentu. Jadi bisa diibaratkan *identifier* dari suatu karakter. Semua karakter ada *code point*-nya, termasuk huruf A, B, C, maupun karakter lainnya (angka, tulisan romawi, symbol, dll).

Cara penulisan karakter unicode sendiri bisa dengan langsung menuliskan karakternya, atau bisa juga dengan menuliskan *code point* dalam notasi tertentu.

- Contoh penulisan text dengan langsung menuliskan karakternya:

```
message = "🍌🍌🍌🍌 😊"
print(message)
# output → 🍌🍌🍌🍌 😊
```

- Menggunakan notasi special character `\uXXXX`, dimana `XXXX` diisi dengan *code point* dalam encoding 16-bit.

```
message = "\uC548\uB155\uD558\uC138uC694"
print(message)
# output → 🍌🍌🍌🍌
```

- *Code point* 16-bit `C548` merepresentasikan karakter 🍌
- *Code point* 16-bit `B155` merepresentasikan karakter 🍌
- *Code point* 16-bit `D558` merepresentasikan karakter 🍌
- *Code point* 16-bit `C548` merepresentasikan karakter 🍌
- *Code point* 16-bit `C694` merepresentasikan karakter 🍌

Untuk memunculkan emoji menggunakan kode encoding 16-bit butuh tambahan effort karena *code point* emoji tidak cukup jika direpresentasikan oleh *code point* yang lebarnya hanya 16-bit.

- Menggunakan notasi special character `\Uxxxxxxxx`, dimana `xxxxxxxx` diisi *code point* dalam encoding 32-bit.

```
message = "\U0000C548\U0000B155\U0000D558\U0000C138\U0000C694 \U0001F600"
print(message)
```

- Code point 32-bit `0000C548` merepresentasikan karakter `◇`
 - Code point 32-bit `0000B155` merepresentasikan karakter `◇`
 - Code point 32-bit `0000D558` merepresentasikan karakter `◇`
 - Code point 32-bit `0000C138` merepresentasikan karakter `◇`
 - Code point 32-bit `0000C694` merepresentasikan karakter `◇`
 - Code point 32-bit `0001F600` merepresentasikan emoji `😊`
- Atau menggunakan notasi special character `\N{NAME}`, dimana `NAME` diisi dengan nama karakter unicode dalam huruf besar.

```
message = "\N{HANGUL SYLLABLE AN}\N{HANGUL SYLLABLE NYEONG} \N{GRINNING FACE}"
print(message)
# output → ◇◇ 😊
```

- Nama karakter Unicode `HANGUL SYLLABLE AN` merepresentasikan karakter `◇`
- Nama karakter Unicode `HANGUL SYLLABLE NYEONG` merepresentasikan karakter `◇`
- Nama karakter Unicode `GRINNING FACE` merepresentasikan emoji `😊`

Salah satu website yang berguna untuk mencari informasi nama dan code point karakter Unicode:
<https://www.compart.com/en/unicode/>

A.18.2. Fungsi utilitas pada *Unicode*

● Fungsi `ord()`

Fungsi `ord()` digunakan untuk mengambil nilai code point dari suatu karakter. Nilai baliknya adalah numerik berbasis desimal.

```
str = "N"
codePoint = ord(str)
print(f'code point of {str} in decimal: {codePoint}')
# output → code point of N in decimal: 78

str = "◇"
codePoint = ord(str)
print(f'code point of {str} in decimal: {codePoint}')
# output → code point of ◇ in decimal: 50504
```

Untuk menampilkan code point dalam notasi hexadesimal, cukup bungkus menggunakan fungsi `hex()`.

```
str = "◇"
codePoint = ord(str)

print(f'code point of {str} in decimal: {codePoint}')
# output → code point of ◇ in decimal: 50504
```

Bisa dilihat dari program di atas, unicode code point dari karakter `📄` dalam bentuk hexadesimal adalah `c548`. Jika dicek pada praktek sebelumnya, kode hexadesimal yang sama kita gunakan juga dalam penulisan karakter unicode menggunakan notasi `\uXXXX` (yaitu `\uc548`).

● Fungsi `chr()`

Fungsi `chr()` adalah kebalikan dari fungsi `ord()`, kegunaannya adalah untuk menampilkan string sesuai code point.

Pada contoh dibawah ini fungsi `chr()` digunakan untuk memunculkan karakter dengan code point desimal `50504` dan juga hexadesimal `C548`, yang keduanya adalah merepresentasikan karakter yang sama, yaitu `📄`.

```
codePoint = chr(50504)
print(codePoint)
# output → 📄

codePoint = chr(0xC548)
print(codePoint)
# output → 📄
```

Catatan chapter 📄

● Source code praktik

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

● Referensi

- <https://docs.python.org/3/howto/unicode.html#:~:text=Python's%20string%20type%20uses%20the,character%20its%20own%20unique%20code.>
 - <https://docs.python.org/3/howto/unicode.html?highlight=unicode%20howto#the-string-type>
-