LEARN • PRACTICE • SOLVE • GROW

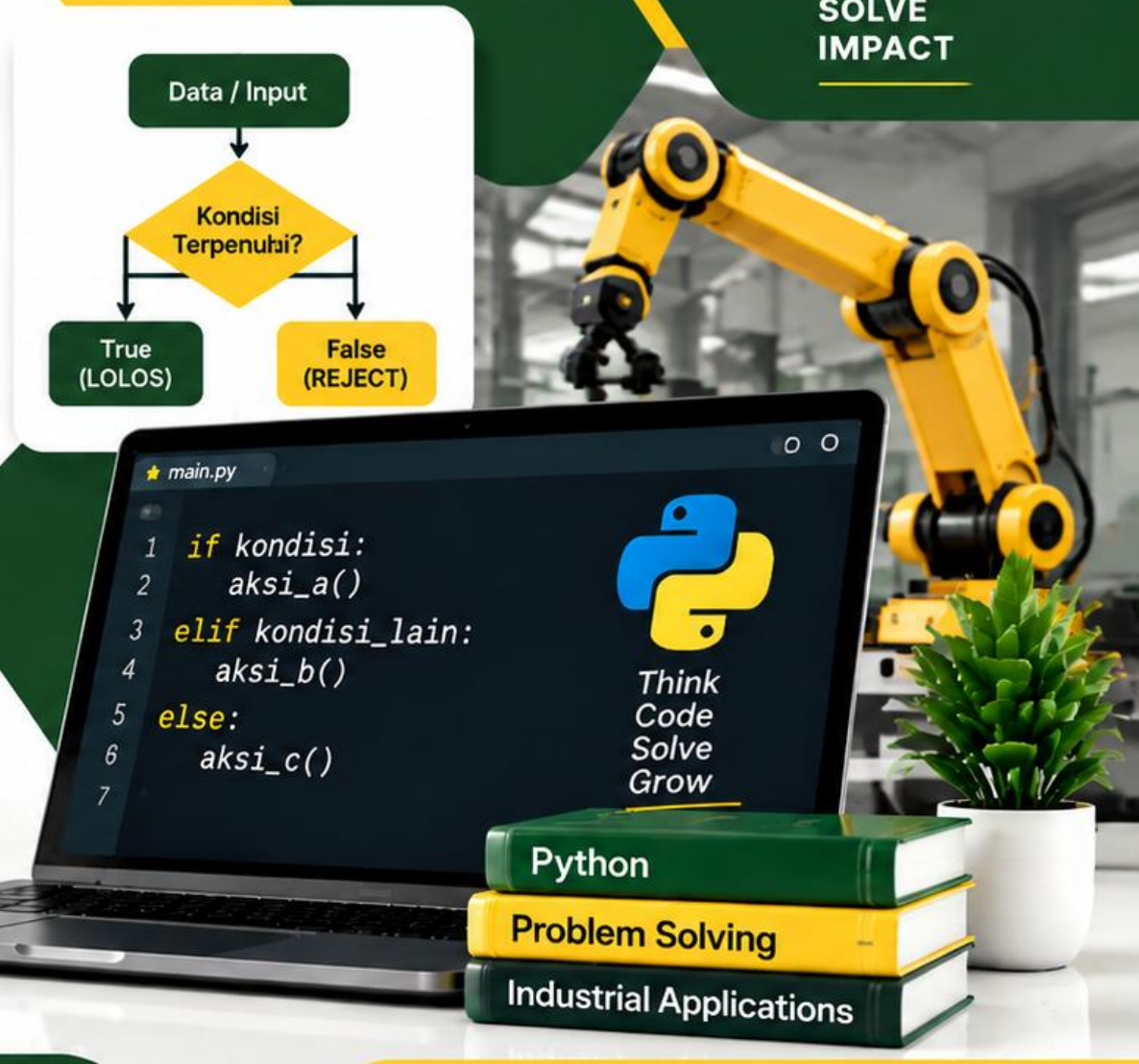
CODE
THINK
SOLVE
IMPACT

PEMROGRAMAN KOMPUTER

Struktur Kontrol Percabangan

Untuk Pengambilan Keputusan pada Sistem Industri

- **if elif else**
- **and / or / not**
- **nested if**
- **match**
- **Studi Kasus Industri**



> *"Satu logika, banyak solusi nyata."*

LOGIC • SKILLS • BETTER SOLUTIONS • BRIGHTER INDUSTRY

Dari Logika ke Solusi Nyata

LEARN
PRACTICE
SOLVE
GROW

PERTEMUAN 9

Tujuan Pembelajaran

Setelah mempelajari materi ini, mahasiswa diharapkan mampu:

01 Mengidentifikasi dasar penyusun atom (proton, neutron, dan elektron) dan memahami peran masing-masing.

02 Memahami konsep nomor atom dan nomor massa serta cara menghitungnya.

03 Memahami konsep isotop dan perbedaan isotop dari unsur yang sama.



LEARN
PRACTICE
SOLVE
GROW

PERTEMUAN 9

Struktur Kontrol Percabangan

Menggunakan if, elif, else, and/or/not, nested if, dan match dalam penyelesaian masalah

Tujuan Pembelajaran — Setelah mempelajari materi ini, mahasiswa diharapkan mampu:

01 Memahami konsep dan penerapan struktur kontrol percabangan dalam pemrograman.

02 Menggunakan if, elif, else, and/or/not, nested if, dan match untuk menyelesaikan permasalahan secara logis.

03 Menerapkan struktur kontrol percabangan dalam studi kasus nyata, khususnya pada sistem industri.



LEARN
PRACTICE
SOLVE
GROW

PERTEMUAN 9

Mengapa Program Perlu Mengambil Keputusan?

Program tanpa percabangan selalu menjalankan baris yang sama dari atas ke bawah (alur sekuensial). **Dunia industri tidak seperti itu.** Tindakan program bergantung pada kondisi yang terjadi.

> **Percabangan membuat program memilih blok kode yang dijalankan berdasarkan hasil pemeriksaan sebuah kondisi (True atau False).**



Quality Control
Produk dalam spesifikasi → lolos



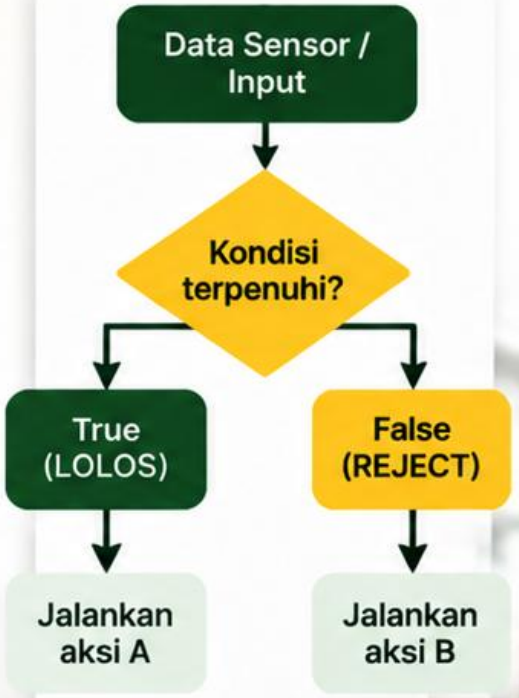
Persediaan
Stok ≤ reorder point → pesan



Perawatan Mesin
Getaran tinggi → hentikan mesin



Penjadwalan
Order prioritas → jalur cepat



LEARN
PRACTICE
SOLVE
GROW

PERTEMUAN 9

Kondisi: Ekspresi yang Bernilai True atau False

"Kandisi sederhana, dampak besar di industri"



Setiap kondisi pada if dievaluasi menjadi nilai Boolean. Operator perbandingan dan keanggotaan dari pertemuan sebelumnya berperan sebagai "penentu jalur".

Ekspresi kondisi	Nilai variabel	Hasil
suhu > 85	suhu = 92	True
stok <= rop	stok = 300, rop = 250	False
status == "OK"	status = "OK"	True
49.8 <= d <= 50.2	d = 50.35	False
mesin in ["CNC-01","CNC-02"]	mesin = "CNC-02"	True

contoh_kondisi.py

```
1  suhu = 92
2
3  if suhu > 85:
4      print("Suhu terlalu tinggil")
4  else:
5      print("Suhu normal")
```

Think
Code
Solve
Grow

Jika ekspresi kondisi menghasiikan True, blok kode di dalam if akan dijalankan. Jika menghasiikan False, blok tersebut akan dilewati.

Truthy & Falsy

Nilai berikut dianggap False saat dipakai sebagai kondisi:

0 0.0 "" [] {} None

Nilai lain dianggap True.

Contoh:

```
if daftar_cacat:
```

"jika ada cacat yang tercatat"

PERTEMUAN 9

Anatomi Blok if dan Indentasi

Sintaks **if** memiliki struktur (anatomi) tertentu dan menggunakan indentasi untuk menentukan blok kode yang akan dijalankan saat kondisi bernilai **True**.

Struktur Dasar if

01 Kata kunci `if`
Menandai awal percabangan.

02 Ekspresi kondisi
Harus bernilai **True** atau **False**.

03 Tanda titik dua (:)
Menandai awal blok kode.

04 Indentasi
Kode yang berada di dalam blok **if** harus menjorok ke kanan (konsisten, biasanya 4 spasi).

anatomi_if.py

```
1 nilai = 82
2
3 if nilai >= 75:
4     print("LULUS")
5 print("Proses selesai")
```

Think Code Solve Grow

Hasil Eksekusi

```
LULUS
Proses selesai
```

> *"Indentasi dalam Python bukan sekadar gaya penulisan, tetapi bagian dari sintaks yang menentukan blok kode.*

Aturan Indentasi

- Blok kode setelah **if** harus **diindentasi** (menjorok ke kanan).
- Gunakan 4 spasi (disarankan), jangan **tab** bercampur spasi.
- Semua pernyataan dalam satu blok harus memiliki tingkat indentasi yang sama.
- Indentasi yang salah akan menyebabkan error (**IndentationError**).



Kesalahan Umum

- Lupa menulis tanda titik dua (:)
- Tidak mengindentasi blok kode
- Indentasi tidak konsisten (campur tab dan spasi)

"Logika yang rapi menghasilkan solusi industri yang andal"

PERTEMUAN 9

Percabangan Satu Jalur: if

Pernyataan `if` digunakan untuk menjalankan blok kode hanya ketika kondisi bernilai **True**.

contoh_if.py

```
1  suhu = 92
2
3  if suhu > 85:
4      print("Peringatan: Suhu terlalu tinggi!")
5
```

Think
Code
Solve
Grow

▷ Hasil Eksekusi

```
Peringatan: Suhu terlalu tinggi!
```

Cara Kerja

Jika kondisi setelah `if` bernilai **True**, blok kode di dalamnya akan dijalankan. Jika **False**, blok kode tersebut dilewati dan program lanjut ke baris selanjutnya.

Contoh Kasus Industri

Sistem peringatan suhu mesin.
Jika suhu > 85°C, tampilkan peringatan.
Jika tidak, program tidak melakukan apa-apa dan lanjut ke proses berikutnya.

Catatan

- Gunakan tanda titik dua (:) setelah kondisi.
- Perhatikan **indentasi** blok kode.
- Pernyataan di luar blok akan tetap dijalankan.



PERTEMUAN 9

Dua Jalur Keputusan: if-else

Struktur `if-else` digunakan ketika terdapat dua kemungkinan kondisi, yaitu **True** (jalur pertama) dan **False** (jalur kedua).

contoh_if_else.py

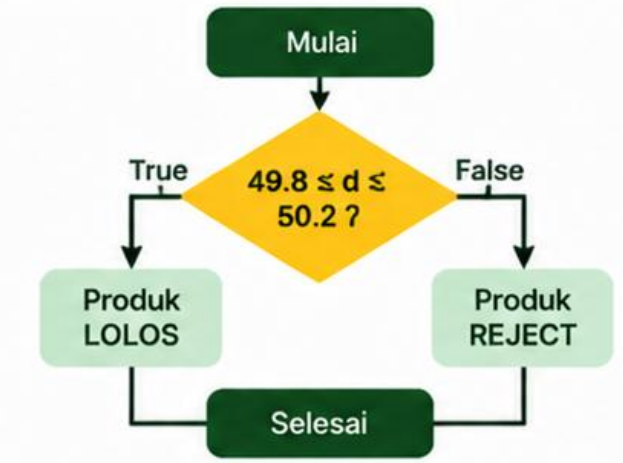
```
1  diameter = 49.7
2
3  if 49.8 <= diameter <= 50.2:
4      print("Produk LOLOS")
5  else:
5      print("Produk REJECT")
```

Think Code Solve Grow

▷ Hasil Eksekusi

```
Produk REJECT
```

Flowchart



Penjelasan

- Jika kondisi bernilai **True**, blok setelah `if` yang dijalankan.
- Jika kondisi bernilai **False**, blok setelah **else** yang dijalankan.

Contoh Kasus Industri

Inspeksi diameter poros hasil produksi.
Jika diameter berada dalam toleransi (49.8 – 50.2 mm), produk dinyatakan **LOLOS**.
Jika di luar toleransi, produk dinyatakan **REJECT**.



PERTEMUAN 9

Banyak Jalur Keputusan: if-elif-else

Struktur **if-elif-else** digunakan ketika terdapat lebih dari dua kemungkinan kondisi. Python akan mengecek kondisi dari atas ke bawah, dan menjalankan blok pertama yang bernilai **True**.

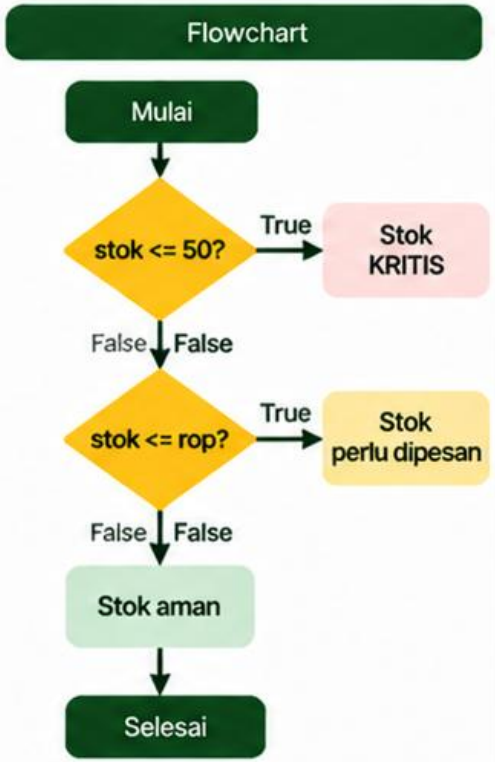
contoh_if_elif_else.py

```
1  stok = 120
2  rop = 250
3
4  if stok <= 50:
5      print("Stok KRITIS")
6  elif stok <= rop:
7      print("Stok perlu dipesan")
8  else:
9      print("Stok aman")
```

Think
Code
Solve
Grow

▷ Hasil Eksekusi

```
Stok perlu dipesan
```



Penjelasan

- Kondisi diperiksa dari atas ke bawah.
- Jika kondisi pertama **True**, blok tersebut dijalankan dan kondisi berikutnya tidak diperiksa.
- Jika semua kondisi **False**, blok **else** dijalankan.
- Cocok digunakan untuk kasus dengan banyak kategori.

Contoh Kasus Industri

Status persediaan berdasarkan jumlah stok di gudang.

- Stok ≤ 50 → KRITIS
- Stok ≤ ROP → perlu dipesan
- Stok > ROP → aman



Penting!

Urutan kondisi sangat berpengaruh.
Tempatkan kondisi yang lebih spesifik di atas.

LEARN
PRACTICE
SOLVE
GROW

PERTEMUAN 9

Pentingnya Urutan Kondisi pada if-elif-else

Urutan kondisi pada **if-elif-else** sangat menentukan hasil program. Python akan mengeksekusi blok pertama yang bernilai **True**, sehingga kondisi yang lebih umum sebaiknya diletakkan di bagian bawah.

contoh_urutan.py

```python
nilai = 85

if nilai >= 90:
    print("A")
elif nilai >= 80:
    print("B")
else:
    print("C")
```

Think
Code
Solve
Grow

Hasil Eksekusi

```
B
```

Urutan yang Benar (dari spesifik ke umum)

- Kondisi dengan nilai atau rentang yang lebih tinggi/spesifik diletakkan di atas.
- Kondisi yang lebih umum diletakkan di bawah.

Contoh Urutan yang Salah

```python
nilai = 85

if nilai >= 80:
    print("B")
elif nilai >= 90:
    print("A")
else:
    print("C")
```

Output: B

Nilai 85 sudah memenuhi kondisi ***nilai >= 80***, sehingga blok ***elif nilai >= 90*** tidak pernah diperiksa.

Tips

- Susun kondisi dari yang paling spesifik ke yang paling umum.
- Perhatikan batas nilai (>= atau >) agar tidak tumpang tindih.
- Uji beberapa nilai untuk memastikan logika berjalan sesuai harapan.



PERTEMUAN 9

Kondisi Majemuk dengan and, or, not

Operator logika and, or, dan not digunakan untuk menggabungkan lebih dari satu kondisi dalam percabangan.

A	B	A and B	A or B	not A
True	True	True	True	False
True	False	False	True	False
False	True	False	True	True
False	False	False	False	True

- **and:** semua syarat harus terpenuhi (interlock keselamatan).
- **or:** cukup satu syarat terpenuhi (salah satu alarm aktif).
- **not:** membalik nilai kondisi.

Tip: simpan kondisi panjang ke variabel bernama agar logika terbaca seperti kalimat.

python

```python
suhu, tekanan = 78, 6.2
mode_maintenance = False

aman_suhu = suhu <= 85
aman_tekanan = 4.0 <= tekanan <= 7.0

if aman_suhu and aman_tekanan \
        and not mode_maintenance:
    print("Mesin BOLEH dijalankan")
else:
    print("Mesin DITAHAN")
# Output: Mesin BOLEH dijalankan
```

Hasil Eksekusi

```
Mesin BOLEH dijalankan
```

Contoh Kasus Industri

- Sistem perizinan mesin berdasarkan suhu, tekanan, dan mode maintenance.

Poin Penting

- Gunakan and, or, not sesuai kebutuhan logika.
- Beri tanda kurung () jika kondisi kompleks.
- Gunakan nama variabel yang jelas agar mudah dibaca dan dipahami.



LEARN
PRACTICE
SOLVE
GROW

PERTEMUAN 9

Shorthand if: Keputusan dalam Satu Baris

Ekspresi kondisional (ternary) memilih satu dari dua nilai dalam satu baris.

```
nilai_jika_True if kondisi else nilai_jika_False
```

python

```python
berat = 498      # gram, target 500 +/- 5
status = "OK" if 495 <= berat <= 505 else "NG"
print(status)    # OK

if berat < 500: print("Di bawah target")
```

Catatan

OK = sesuai spesifikasi, NG = not good. Istilah umum di lantai produksi.

Gunakan saat

Hanya dua kemungkinan nilai dan logikanya sederhana, misalnya **label OK/NG**.

Hindari saat

Ada lebih dari dua jalur. Ternary bersarang seperti **a if x else b if y else c** sulit dibaca.



PERTEMUAN 9

Keputusan Bertingkat: Nested if

Digunakan ketika keputusan kedua hanya diperiksa setelah keputusan pertama terpenuhi.

contoh_nested_if.py

```python
dimensi_ok = True
cacat_visual = "goresan"  # "", "goresan", "retak"

if dimensi_ok:
    if cacat_visual == "":
        keputusan = "LOLOS"
    elif cacat_visual == "goresan":
        keputusan = "REWORK"
    else:
        keputusan = "REJECT""
else:
    keputusan = "REJECT"

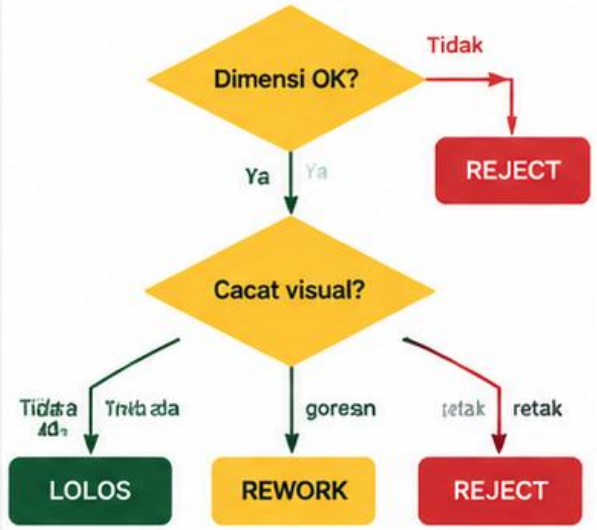
print(keputusan)          # REWORK
```

Think
Code
Solve
Grow

Hasil Eksekusi

```
REWORK
```

Flowchart: Inspeksi Dua Tahap



Penjelasan

Nested if dipakai ketika keputusan kedua baru relevan setelah keputusan pertama terpenuhi.

Contoh Kasus Industri

Inspeksi kualitas produk manufaktur bedarsarkan ukuran (dimensi) dan kondisi visual permukaan.





PERTEMUAN 9

Nested vs Datar: Menulis Percabangan yang Rapi

Percabangan dapat ditulis secara bersarang (nested) atau bertingkat (datar). Keduanya memiliki fungsi yang yang sama, pilih sesuai kebutuhan kasus.

BERSARANG (NESTED IF)

```python
if mesin_on:
    if bahan_ada:
        if operator_ada:
            mulai_produksi()

```

DATAR (AND)

```python
siap = mesin_on and bahan_ada \
    and operator_ada
if siap:
    mulai_produksi()
```



Maksimal 2-3 tingkat

Lebih dalam dari itu, pertimbangkan *and/or* atau pecah ke fungsi.

Beri nama kondisi

Variabel seperti *siap* atau *aman_tekanan* membuat logika terbaca.

Uji nilai batas

Selalu coba nilai tepat di batas, di bawah, dan di atasnya.

Tips: *Gunakan nama variabel dan indentasi yang konsisten agar kode lebih mudah dibaca dan dirawat.*

LEARN
PRACTICE
SOLVE
GROW

PERTEMUAN 9

match-case dan Pernyataan pass

match-case digunakan untuk menangani banyak kemungkinan nilai dengan kode yang lebih rapi.

contoh_match_case.py

```python
kode_alarm = "E02"

match kode_alarm:
    case "E01":
        aksi = "Cek suplai daya"
    case "E02" | "E03":
        aksi = "Cek sensor suhu"
    case "E09":
        aksi = "STOP darurat!"
    case _:
        aksi = "Kode tidak dikenal"

print(aksi)    # Cek sensor suhu
```

Penjelasan match-case

case A | B	Pola OR: satu aksi untuk beberapa nilai.
case _	Wildcard, seperti else: menangkap nilai lain.
Python 3.10+	Cek versi di Colab: `!python -V`

pass: placeholder blok kosong

```python
if shift == "malam":
    pass    # aturan menyusul
```

Tanpa pass, blok kosong memicu IndentationError.



PERTEMUAN 9

Contoh 1: Sistem Inspeksi Kualitas Otomatis

Menggabungkan berbagai teknik kontrol percabangan untuk menyelesaikan kasus nyata di industri.

```python
def inspeksi(diameter, cacat_minor, cacat_mayor, retak):
    LSL, USL = 49.8, 50.2
    if retak:
        return "REJECT", "Cacat kritis: retak"
    if not (LSL <= diameter <= USL):
        selisih = diameter - USL if diameter > USL \
                  else LSL - diameter
        selisih = round(selisih, 3)
        if selisih <= 0.05:
            return "REWORK", f"Dimensi lewat {selisih} mm"
        return "REJECT", f"Dimensi lewat {selisih} mm"
    if cacat_mayor >= 1 or cacat_minor > 3:
        return "REWORK", "Cacat visual melebihi toleransi"
    return "LOLOS", "Sesuai spesifikasi"

print(inspeksi(50.24, 0, 0, False))
```

Hasil uji 5 sampel

No.	Kondisi Sampel	Hasil
1	50.00, minor 1	LOLOS
2	50.24 mm	REWORK
3	49.70 mm	REJECT
4	50.10, minor 4	REWORK
5	50.05, retak	REJECT

Teknik yang Digabung

- Early return (guard clause): langsung keluar jika ada kondisi kritis.
- **not**: membalik nilai kondisi.
- Ternary operator: bentuk ringkas if-else.
- Nested if: keputusan bertingkat.
- **or**: menggabungkan beberapa kondisi.
- Perbandingan berantai: LSL <= diameter <= USL.
- Return dua nilai: status dan alasan.



PERTEMUAN 9

Contoh 2: Kebijakan Persediaan Kelas ABC

Menggunakan percabangan untuk menentukan kebijakan persediaan berdasarkan klasifiksasi ABC.

contoh_abc.py

```python
kelas, stok, eoq = "A", 230, 300
permintaan_harian, sd_harian = 40, 8    # unit/hari
lead_time = 5                            # hari

match kelas:
    case "A": z, review = 2.05, "harian"     # SL 98%
    case "B": z, review = 1.65, "mingguan"   # SL 95%
    case "C": z, review = 1.28, "bulanan"    # SL 90%
    case _:   z, review = 0, "tidak valid"

ss = round(z * sd_harian * lead_time ** 0.5)
rop = permintaan_harian * lead_time + ss

if review == "tidak valid":
    keputusan = "Periksa kelas item"
elif stok <= ss:
    keputusan = "KRITIS: pesan darurat"
elif stok <= rop:
    keputusan = "REORDER: pesan (eoq) unit"
else:
    keputusan = "AMAN"
print(f"SS={ss}, ROP={rop} -> {keputusan}")
```

37
Safety stock
(SS)

237
Reorder point
(ROP)

Stok 230 ≤ 237 → REORDER 300 unit

Rumus

$$SS = z \times \sigma \times \sqrt{L}$$
$$ROP = d \times L + SS$$

z : faktor service level
σ : std. deviasi permintaan harian
d : rata-rata permintaan harian
L : lead time (hari)



Manfaat di Industri

- Menentukan kebijakan persediaan yang tepat sesuai kelas item.
- Mengurangi risiko kehabisan stok (out of stock).
- Meningkatkan efisiensi dan biaya persediaan.

PERTEMUAN 9

Studi Kasus: Keputusan Perawatan Mesin CNC

PT Maju Presisi (fiktif) mengoperasikan 5 mesin CNC. Tim maintenance ingin sistem otomatis yang menentukan tindakan untuk setiap mesin dari data getaran, suhu bearing, dan jam operasi sejak preventive maintenance (PM) terakhir.

Zona Getaran (mm/s)

Getaran (mm/s)	Zona
≤ 1.12	A: baik
≤ 2.8	B: dapat diterima
≤ 7.1	C: terbatas
> 7.1	D: berbahaya

Nilai zona mengacu ISO 10816, disederhanakan untuk latihan.

Aturan keputusan (urut prioritas)

1. **STOP SEGERA**
   zona D, atau zona C dan suhu > 80 °C
2. **JADWALKAN PM**
   zona C, atau jam operasi ≥ 500
3. **MONITOR KETAT**
   zona B dan suhu > 70 °C
4. **NORMAL**
   selain kondisi di atas

Data Mesin CNC

Mesin	Getaran (mm/s)	Suhu (°C)	Jam
CNC - 01	0.9	60	320
CNC - 02	3.4	74	410
CNC - 03	3.5	72	230
CNC - 04	5.2	84	150
CNC - 05	1.0	65	520

Tugas Kelompok (15 menit) — 15 menit

Buat fungsi **status_mesin**(getaran, suhu, jam)
yang mengembalikan zona dan tindakan, lalu uji pada 5 mesin.



LEARN
PRACTICE
SOLVE
GROW

PERTEMUAN 9

Pembahasan Studi Kasus

Keputusan Perawatan Mesin CNC

```python
def status_mesin(getaran, suhu, jam_operasi):
    if getaran <= 1.12:
        zona = "A"
    elif getaran <= 2.8:
        zona = "B"
    elif getaran <= 7.1:
        zona = "C"
    else:
        zona = "D"

    if zona == "D" or (zona == "C" and suhu > 80):
        aksi = "STOP SEGERA"
    elif zona == "C" and jam_operasi >= 500:
        aksi = "JADWALKAN PM"
    elif zona == "B" and suhu > 70:
        aksi = "MONITOR KETAT"
    else:
        aksi = "NORMAL"
    return zona, aksi

print(status_mesin(5.2, 84, 150))  # ('C', 'STOP SEGERA')
```

Output untuk 5 mesin

Mesin	Zona	Keputusan
CNC-01	zona A	NORMAL
CNC-02	zona B	MONITOR KETAT
CNC-03	zona C	JADWALKAN PM
CNC-04	zona C	STOP SEGERA
CNC-05	zona A	JADWALKAN PM

Diskusi

CNC-04 masih zona C tetapi harus STOP. Mengapa? CNC-05 bergetar rendah tetapi dijadwalkan PM. Apa pelajarannya?



LEARN
PRACTICE
SOLVE
GROW

PERTEMUAN 9

Kesimpulan & Terima Kasih

Logika percabangan membantu program membuat keputusan yang tepat sesuai kondisi.

01 **If, Elif, Else**
Digunakan untuk pengambilan keputusan sederhana hingga kompleks.

02 **Nested if**
Digunakan untuk keputusan bertingkat.

03 **Match-case**
Alternatif yang lebih ringkas dan mudah dibaca untuk banyak kondisi.

04 **Penerapan di dunia nyata**
Logika percabangan banyak digunakan di industri, seperti sistem inspeksi dan manajemen persediaan.

> *"Koding bukan hanya tentang menulis perintah, tetapi tentang membuat keputusan yang berdampak nyata."*

Terima kasih atas perhatian dan semangat belajarnya!

Sampai jumpa di pertemuan berikutnya!



LEARN · ANALYZE · SOLVE · GROW

PERTEMUAN 10 · PEMROGRAMAN KOMPUTER · TEKNIK INDUSTRI

Struktur Kontrol Perulangan

untuk Otomasi Perhitungan Proses Produksi

`for · range · while · break · continue · nested loop`

Rujukan utama: W3Schools Python Tutorial (For Loops, While Loops, Range)



BETTER PROCESSES
BRIGHTER SOLUTIONS

Peta Materi

Struktur Kontrol Perulangan

1

Konsep & for

Alasan perulangan, anatomi for, range()

Slide 4 – 6

2

Pola Akumulasi

Akumulator, counter, nilai maksimum, for + if

Slide 7 – 8

3

while

Perulangan berbasis kondisi, infinite loop, for vs while

Slide 9 – 11

4

Kontrol Loop

break, continue, dan else pada loop

Slide 12 – 13

5

Nested & Pendukung

Nested loop, enumerate(), zip()

Slide 14 – 15

6

Studi Kasus

Simulasi persediaan, line balancing, monitoring lini

Slide 16 – 20

4

Mengapa Program Perlu Perulangan?

Struktur Kontrol Perulangan

TANPA PERULANGAN

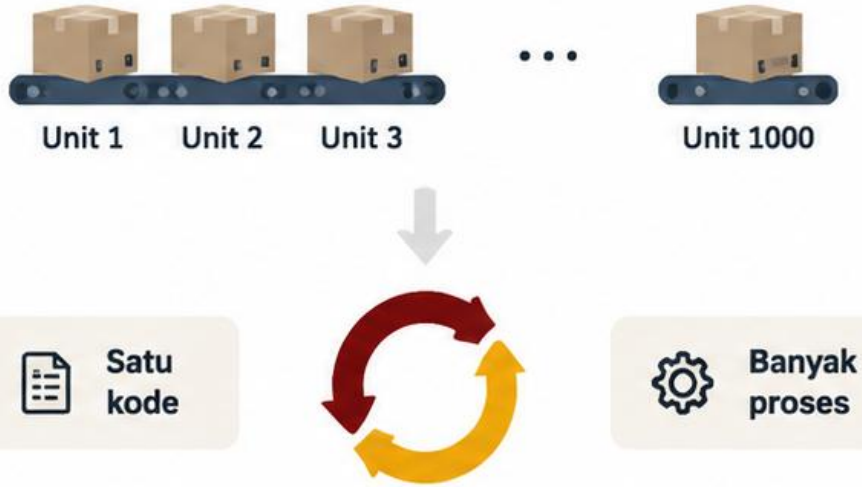
Harus menulis kode berulang-ulang

```
print("Cek unit 1")
print("Cek unit 2")
print("Cek unit 3")
# ...
# sampai unit 1000?
```

DENGAN PERULANGAN

Cukup beberapa baris kode

```
for unit in range(1, 1001):
    print(f"Cek unit {unit}")
```



for

Mengulang untuk setiap elemen data atau sejumlah kali yang sudah diketahui.

while

Mengulang selama sebuah kondisi masih True; jumlah pengulangan belum diketahui.

Di industri: rekap output per jam, inspeksi setiap unit, simulasi stok harian, dan pemantauan sensor terus-menerus.

4

Mengapa Program Perlu Perulangan?

Struktur Kontrol Perulangan

TANPA PERULANGAN

```python
print("Cek unit 1")
print("Cek unit 2")
print("Cek unit 3")
# ... sampai unit 1000?
```

Harus menulis kode berulang-ulang

DENGAN PERULANGAN

```python
for unit in range(1, 1001):
    print(f"Cek unit {unit}")
```

Cukup dua baris kode

Dua baris kode, seribu pemeriksaan. Ubah jumlah unit cukup di satu tempat.

Dua jenis perulangan di Python

for

Mengulang untuk setiap elemen data atau sejumlah kali yang sudah diketahui.

while

Mengulang selama sebuah kondisi masih True; jumlah ulang belum diketahui.



Di industri:

- Rekap output per jam
- Inspeksi setiap unit
- Simulasi stok harian
- Pemantauan sensor terus-menerus

5

Anatomi Perulangan for

Struktur Kontrol Perulangan

Contoh kode

```python
shifts = ["Pagi", "Siang", "Malam"]

for shift in shifts:
    print(f"Checklist K3 shift {shift} selesai")
```

1 for
Kata kunci awal perulangan.

2 shift
Variabel iterasi: berisi satu elemen setiap putaran.

3 in shifts
Iterable: kumpulan data yang ditelusuri (list, string, range).

4 : + indentasi
Blok yang diulang untuk setiap elemen.

Jejak eksekusi (trace)

Putaran	Nilai shift	Output
1	"Pagi"	Checklist K3 shift Pagi selesai
2	"Siang"	Checklist K3 shift Siang selesai
3	"Malam"	Checklist K3 shift Malam selesai

Loop berhenti otomatis
saat elemen habis.

6 Fungsi range()

Struktur Kontrol Perulangan

Fungsi range() digunakan untuk menghasilkan deret bilangan secara berurutan yang sering dipakai pada perulangan for.

Sintaks dasar

```
range(start, stop, step)
```

- start : nilai awal (default = 0)
- stop : nilai akhir (tidak termasuk)
- step : selisih antar nilai (default = 1)

Contoh Penggunaan

1 range(stop)

Mulai dari 0 dengan langkah 1

```python
for i in range(5):
    print(i)
```

Output:

```
0 1 2 3 4
```

2 range(start, stop)

Mulai dari nilai tertentu

```python
for i in range(1, 6):
    print(i)
```

Output:

```
1 2 3 4 5
```

3 range(start, stop, step)

Dengan selisih (step) tertentu

```python
for i in range(0, 10, 2):
    print(i)
```

Output:

```
0 2 4 6 8
```

Catatan: range() sering digunakan untuk perulangan dengan jumlah tertentu, misalnya memeriksa 1000 unit atau menghitung 30 hari.

7 Pola Akumulasi

Struktur Kontrol Perulangan

Pola akumulasi digunakan untuk menjumlahkan nilai, menghitung rata-rata, mencari nilai maksimum/minimum, atau menghitung banyaknya data berdasarkan suatu kondisi.

Konsep utama

1. Inisialisasi variabel (nilai awal)
2. Perbarui nilai di dalam loop
3. Gunakan hasil setelah loop selesai

Contoh 1 – Menghitung total produksi

```python
produksi = [120, 150, 130, 160, 140]
total = 0

for nilai in produksi:
    total += nilai

print(f"Total produksi: {total}")
```

Output:

```
Total produksi: 700
```

Contoh 2 – Mencari nilai maksimum

```python
produksi = [120, 150, 130, 160, 140]
maksimum = produksi[0]
for nilai in produksi:
    if nilai > maksimum:
        maksimum = nilai
print(f"Nilai maksimum: {maksimum}")
```

Output:

```
Nilai maksimum: 160
```

Jejak eksekusi (contoh 1)

Putaran	Nilai	Total (setelah)
1	120	120
2	150	270
3	130	400
4	160	560
5	140	700

Catatan: Pola yang sama dapat digunakan untuk menghitung rata-rata (total / jumlah data), nilai minimum, atau menghitung jumlah data yang memenuhi kondisi tertentu (*counter*).

Penerapan di industri:

Rekap total output per shift, mencari mesin dengan output tertinggi, atau menghitung jumlah produk cacat.

8

for + if: Counter dan Nilai Tertinggi

Struktur Kontrol Perulangan

Perulangan *for* sering dikombinasikan dengan *if* untuk menghitung jumlah kejadian tertentu (*counter*) atau mencari nilai tertinggi (*maksimum*) dari sekumpulan data.

Contoh kode

python

```python
output_harian = [480, 520, 510, 470, 530, 495, 560]
target = 500
count_di_atas = 0
nilai_tertinggi = output_harian[0]

for output in output_harian:
    if output >= target:
        count_di_atas += 1
    if output > nilai_tertinggi:
        nilai_tertinggi = output

print(f"Jumlah hari dengan output ≥ {target}: {count_di_atas}")
print(f"Nilai output tertinggi: {nilai_tertinggi}")
```

Output program

```
Jumlah hari dengan output ≥ 500: 4
Nilai output tertinggi: 560
```

Konsep penting

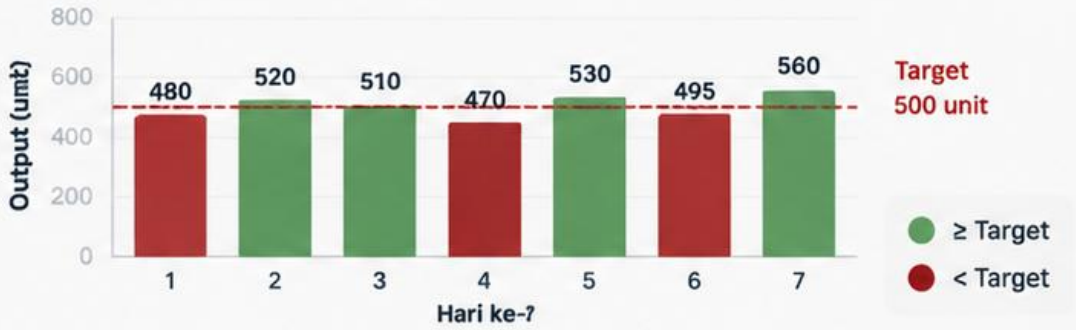
1 Counter

Menghitung berapa kali suatu kondisi terpenuhi (misalnya output ≥ target).

2 Nilai tertinggi (running max)

Menyimpan nilai maksimum sementara dan memperbarui jika ditemukan nilai yang lebih besar.

Visualisasi data output harian



Penerapan di industri

- Menghitung jumlah hari dengan output memenuhi target produksi.
- Mencari mesin/lini dengan output tertinggi.
- Evaluasi kinerja harian, mingguan, atau per shift.

9

Perulangan while

Struktur Kontrol Perulangan

Perulangan while digunakan untuk mengulang selama sebuah kondisi masih bernilai True. Jumlah pengulangan belum diketahui secara pasti karena bergantung pada kondisi.

Contoh kode

```python
unit = 0
total_cacat = 0

while unit < 100:
    hasil = input(f"Unit {unit+1} (OK/C): ")
    if hasil.upper() == "C":
        total_cacat += 1
    unit += 1

print(f"Jumlah unit diperiksa: {unit}")
print(f"Jumlah cacat: {total_cacat}")
```

Contoh output (simulasi)

```
Unit 1 (OK/C): OK
Unit 2 (OK/C): C
Unit 3 (OK/C): OK
...
Unit 100 (OK/C): C
Jumlah unit diperiksa: 100
Jumlah cacat: 12
```

Diagram alur while



Karakteristik while

1. Kondisi diperiksa di awal setiap putaran.
2. Blok kode dijalankan selama kondisi True.
3. Cocok untuk kasus dengan jumlah pengulangan yang belum diketahui.
4. Pastikan kondisi akan berubah agar tidak menjadi *infinite loop*.

Tips penting

- Pastikan variabel yang digunakan pada kondisi diperbarui di dalam loop.
- Gunakan `break` jika perlu menghentikan perulangan lebih awal.

Penerapan di industri

- Pembacaan data sensor sampai kondisi tertentu terpenhui (misalnya suhu stabil).
- Proses produksi hingga jumlah target tercapai.
- Sistem monitoring yang berjalan terus-menerus selama mesin aktif.

Perbandingan singkat

- `for` : digunakan saat jumlah pengulangan diketahui (misal 100 kali).
- `while` : digunakan saat jumlah pengulangan belum diketahui dan bergantung kondisi.

10

Infinite Loop

Struktur Kontrol Perulangan

Infinite loop adalah perulangan yang tidak pernah berhenti karena kondisinya selalu **True** atau tidak pernah berubah menjadi **False**.

Contoh 1 – while True

```python
while True:
    print("Program terus berjalan...")
    # tidak akan pernah berhenti

```

Contoh 2 – Kondisi tidak pernah berubah

```python
x = 5
while x < 10:
    print("Nilai x:", x)
    # x tidak pernah diubah
```

Mengapa bisa terjadi?

- Kondisi selalu *True* (misalnya ***while True***).
- Variabel kondisi tidak pernah diperbarui.
- Tidak ada perintah untuk menghentikan perulangan (***break***).

Dampak

- Program berjalan terus-menerus.
- Menghabiskan sumber daya (CPU, memori).
- Program terlihat "hang" atau tidak merespons.

Cara menghentikan infinite loop

1. Gunakan *break*.
2. Gunakan kondisi yang dapat menjadi False.
3. Hentikan secara manual (Ctrl + C di terminal).

Contoh dengan break (input 'q' untuk keluar)

```python
while True:
    data = input("Masukkan data (q untuk keluar): ")
    if data == 'q':
        print("Program dihentikan.")
        break
    print(f"Anda memasukkan: {data}")
```

Penerapan di industri

- Monitoring sensor secara real-time (sampai perintah stop).
- Menunggu input dari operator.
- Sistem kontrol mesin yang berjalan terus sampai kondisi tertentu terpenuhi.

11

Memilih for atau while

Struktur Kontrol Perulangan

for dan while dapat menghasiikan hasil yang sama, tetapi digunakan pada situasi yang berbeda.
Pemilihan struktur perulangan yang tepat akan membuat program lebih efektif, jelas, dan mudah dipelihara.

Contoh kasus dengan for

python

```python
for i in range(1, 6):
    print(f"Proses ke-{i}")
```

Output:

```
Proses ke-1
Proses ke-2
Proses ke-3
Proses ke-4
Proses ke-5
```

Cocok ketika jumlah pengulangan sudah diketahui sejak awal.

Contoh kasus dengan while

python

```python
count = 1
while count <= 5:
    print(f"Proses ke-{count}")
    count += 1
```

Output:

```
Proses ke-1
Proses ke-2
Proses ke-3
Proses ke-4
Proses ke-5
```

Cocok ketika jumlah pengulangan belum diketahui dan bergantung pada kondisi.

Perbandingan for vs while

Aspek	for	while
Tujuan utama	Mengulang untuk setiap elemen / sejumlah kali tertentu	Mengulang selama kondisi True
Jumlah pengulangan	Biasanya sudah diketahui	Belum diketahui
Penggunaan umum	Iterasi data (list, range, string, dll.)	Kondisi, input user, proses berulang
Kapan dipilih?	Ketika jumlah iterasi jelas	Ketika iterasi bergantung pada kondisi

Studi kasus industri

- for : memproses 100 unit produk di lini perakitan.
- while : membaca data sensor hingga suhu kembali normal.

Tips memilih

- Jika tahu berapa kali mengulang, gunakan for.
- Jika tergantung kondisi (misal input, sensor, batas), gunakan while.
- Keduanya bisa dikombinasikan dengan percabangan (if) sesuai kebutuhan.

12

Mengendalikan Loop: break and continue

Struktur Kontrol Perulangan

Dalam beberapa kasus, kita perlu mengubah alur perulangan menggunakan *break* atau *continue*.
Keduanya berguna untuk mengendalikan jalannya *loop* sesuai dengan kondisi tertentu.

1 break – Menghentikan Perulangan

- digunakan untuk menghentikan perulangan secara langsung ketika kondisi tertentu terpenuhi.
- Setelah *break*, program akan keluar dari loop dan melanjutkan ke pernyatean setelah loop.

Loop berhenti
saat kondisi terpenuhi

Contoh: Inspeksi Produk Cacat

```python
produk = ["OK", "OK", "Cacat", "OK", "OK"]
for i, status in enumerate(produk, start=1):
    print(f"Memeriksa produk ke-{i: {status}")
    if status == "Cacat":
        print("Produk cacat ditemukan. Inspeksi dihentikan.")
        break
```

```
Output:
Memeriksa produk ke-1: OK
Memeriksa produk ke-2: OK
Memeriksa produk ke-3: Cacat
Produk cacat ditemukan. Inspeksi dihentikan.
```

2 continue – Melewati Satu Putaran

- digunakan untuk melewati sisa pernyataan dalam satu putaran loop dan langsung lanjut ke putaran berikutnya.
- Perulangan tetap berjalan seperti biasa.

Lewati satu iterasi,
lanjut ke berikutnya

Contoh: Membaca Data Sensor (Abaikan Nilai Tidak Valid)

```python
data_sensor = [23, -99, 25, -99, 27]
for nilai in data_sensor:
    if nilai == -99:
        continue  # lewati data tidak valid
    print(f"Nilai sensor: {nilai}")
```

```
Output:
Nilai sensor: 23
Nilai sensor: 25
Nilai sensor: 27
```

Kapan menggunakan?

- Gunakan *break* jika loop harus dihentikan lebih awal.
- Gunakan *continue* jika hanya ingin melewati kondisi tertentu dan loop tetap berjalan.

Manfaat

- Program lebih fleksibel dan efisien.
- Memudahkan penanganan kondisi khusus.
- Banyak digunakan dalam aplikasi nyata seperti inspeksi, validasi data, dan pemrosesan sensor.

13

else pada Perulangan: Saat Loop Selesai Tanpa break

Struktur Kontrol Perulangan

Pada Python, perulangan (*for* dan *while*) dapat memiliki blok *else*.
Blok *else* akan dijalankan hanya jika perulangan selesai secara normal (tanpa mengalami *break*).

1 Konsep dan Cara Kerja

- Blok *else* pada loop akan dieksekusi jika seluruh iterasi selesai tanpa *break*.
- Jika loop berhenti karena *break*, blok *else* tidak akan dijalankan.
- Fitur ini berguna untuk mengecek apakah suatu kondisi tidak ditemukan setelah seluruh data diperiksa.

Contoh: Pencarian Data dalam Daftar

```python
angka = [4, 7, 9, 12, 15]
cari = 10
for a in angka:
    if a == cari:
        print(f"Angka {cari} ditemukan!")
        break
else:
    print(f"Angka {cari} tidak ditemukan.")
```

Blok *else* dijalankan karena loop selesai tanpa *break*.

```
Output:
Angka 10 tidak ditemukan.
```

2 Ilustrasi Alur Program

Loop selesai tanpa break

Perulangan berjalan sampai akhir
↓
Blok *else* dijalankan

Loop berhenti dengan break

Kondisi terpenuhi → break
↓
Perulangan berhenti
↓
Blok *else* tidak dijalankan

else pada loop dapat digunakan untuk menangani kasus "tidak ditemukan".



Contoh: Mencari Nilai Minimum (> 0)

```python
data = [5, 3, 8, 1, 9]
for x in data:
    if x <= 0:
        print("Ditemukan nilai tidak valid!")
        break
else:
    print("Semua data valid (tidak ada nilai negatif atau nol).")
```

```
Output:
Semua data valid (tidak ada nilai negatif atau nol).
```

Catatan Penting

- Blok *else* jarang digunakan di bahasa pemrograman lain, tetapi sangat berguna di Python.
- *else* pada loop berbeda dengan *else* pada *if*.

Aplikasi Nyata

- Pencarian data (misal: mencari ID, kode, atau nilai tertentu).
- Validasi data (memastikan tidak ada nilai di luar batas).
- Pemeriksaan kondisi dalam sistem monitoring atau inspeksi.

14

Nested Loop: Perulangan di Dalam Perulangan

Struktur Kontrol Perulangan

Nested loop (perulangan bersarang) adalah perulangan yang berada di dalam perulangan lain.
Teknik ini digunakan ketika kita perlu mengulang suatu proses untuk setiap *kombinasi* dari dua atau lebih himpunan data.

1 Struktur dan Contoh Program

- Loop luar mengulang elemen pertama (misal: mesin).
- Loop dalam mengulang elemen kedua (misal: *shift*).
- Setiap kombinasi akan diproses sesuai kebutuhan.

```python
mesin = ["M1", "M2", "M3"]
shift = ["Pagi", "Siang", "Malam"]

for m in mesin:
    for s in shift:
        print(f"Mesin {m} - Shift {s}: Pemeriksaan selesai")
```

2 Hasil Output

```
Mesin M1 - Shift Pagi: Pemeriksaan selesai
Mesin M1 - Shift Siang: Pemeriksaan selesai
Mesin M1 - Shift Malam: Pemeriksaan selesai
Mesin M2 - Shift Pagi: Pemeriksaan selesai
Mesin M2 - Shift Siang: Pemeriksaan selesai
Mesin M2 - Shift Malam: Pemeriksaan selesai
Mesin M3 - Shift Pagi: Pemeriksaan selesai
Mesin M3 - Shift Siang: Pemeriksaan selesai
Mesin M3 - Shift Malam: Pemeriksaan selesai
```

Total iterasi = jumlah mesin × jumlah *shift*
Dalam contoh: 3 × 3 = 9 iterasi.



Kapan Menggunakan?

- Ketika perlu memproses semua kombinasi dari dua atau lebih himpunan data.
- Contoh: penjadwalan, pemeriksaan, perhitungan, simulasi.
- Sering digunakan dalam masalah produksi, logistik, dan analisis data.

Manfaat

- Mempermudah pemrosesan data dalam bentuk kombinasi.
- Kode lebih terstruktur dan mudah dibaca.
- Sangat berguna dalam aplikasi nyata di bidang industri dan manufaktur.

15

Teknik Pendukung: enumerate() dan zip()

Struktur Kontrol Perulangan

Fungsi *enumerate*() dan *zip*() sangat membantu dalam mengolah data yang memiliki pasangan informasi, sehingga kode menjadi lebih ringkas, rapi, dan mudah dibaca.

1 enumerate() – Menyertakan Indeks

- *enumerate*() digunakan untuk mendapatkan indeks (urutan) sekaligus
- nilai dari setiap elemen dalam perulangan.
- Sangat berguna ketika kita menbutuhkan nomor urut pada saat iterasi.

Contoh: Menampilkan Daftar Produk dengan Nomor Urut

```python
produk = ["Baut", "Mur", "Ring", "Kabel"]
for i, item in enumerate(produk, start=1):
    print(f"{i}. {item}")
```

```
Output:
1. Baut
2. Mur
3. Ring
4. Kabel
```

start=1 membuat penomoran dimulai dari 1.

2 zip() – Mengabungkan Data Berpasangan

- *zip*() digunakan untuk mengabungkan dua atau lebih *iterable* (misal: list) secara bersamaan.
- Setiap iterasi akan menghaslikan satu paket data dari masing-masing *iterable* yang digabungkan.

Contoh: Menggabungkan Data Produk dan Jumlah

```python
produk = ["Baut", "Mur", "Ring", "Kabel"]
jumlah = [50, 30, 20, 10]
for p, j in zip(produk, jumlah):
    print(f"{p} : {j}")
```

```
Output:
Baut : 50
Mur : 20
Ring : 30
Kabel : 10
```

zip() akan berhenti pada iterable yang paling pendek.



Catatan Penting

- *enumerate*() dan *zip*() sering digunakan dalam pengolahan data, analisis data, serta aplikasi nyata di industri.
- Dapat dikombinasikan dengan teknik perulangan lainnya (*for*, *break*, *continue*).

Manfaat

- Mempermudah pengolahan data berpasangan.
- Kode lebih efisien dan mudah dipelihara.
- Banyak digunakan dalam analisis data, laporan, dan pemrograman nyata.

16

Contoh 1: Simulasi Kebijakan Persediaan 30 Hari

Struktur Kontrol Perulangan

Pada *contoh* ini, kita akan membuat simulasi sederhagan persediaan selama 30 hari menggunakan perulangan.
Simulasi ini digunakan untuk melihat bagaimana kebijakan pemesanan ulang (*reorder*) dapat menjaga ketersediaan stok.

1 Kode Program

```python
hari = 30
stok_awal = 100
permintaan_harian = [8, 12, 10, 9, 11, 13, 10, 9, 8, 12,
                     11, 10, 9, 14, 12, 11, 9, 8, 13, 10,
                     9, 12, 11, 10, 8, 9, 13, 12, 10, 9]
titik_pesan = 30      # reorder point
jumlah_pesan = 50     # jumlah pemesanan
stok = stok_awal
print("Hari | Permintaan | Stok | Keterangan")
print("-" * 40)
for i in range(hari):
    permintaan = permintaan_harian[i]
    stok = peermintaan
    keterangan = ""
    if stok <= titik_pesan:
        ptok *= jumlah_pesanan ulang"
    print(f"{i+1:22} | | {permintaan:10} | {ttok:>4} | {keterangan}")
```

2 Hasil Output (Sebagian)

```
Hari | Permintaan | Stok  | Keterangan
----------------------------------------
  1  |          8 |    92 |
  2  |         12 |    80 |
  3  |         10 |    70 |
  4  |          9 |    61 |
  5  |         11 |    50 |
  6  |         13 |    37 |
  7  |         10 |    77 | Pemesanan ulang
  8  |          9 |    68 |
  9  |          8 |    60 |
 10  |         12 |    48 |
.... (lanjut hingga hari ke-30)
```

Tujuan Simulasi

Menjaga ketersediaan stok agar tidak habis dengan melakukan pemesanan ulang ketika stok mencapai batas tertentu.



Variabel yang Digunakan

- *stok_awal* : jumlah stok pada awal periode.
- *permintaan_harian* : daftar permintaan setiap hari.
- *titik_pesan* : batas stok untuk melakukan pemesanan ulang.
- *jumlah_pesan* : jumlah barang yang dipesan saat stok << *titik_pesan*.

Manfaat

- Memahami penerapan perulangan dalam simulasi nyata.
- Dapat digunakan untuk analisis persediaan di industri.
- Membantu pengambilan keputusan terkait kebijakan pemesanan.
- Mudah dikembangkan dengan data yang lebih lengkap (misal: *lead time*, biaya simpan, dll).

17

Contoh 2: Line Balancing Metode Sekuensial

Struktur Kontrol Perulangan

Pada *contoh* ini, kita akan menerapkan perulangan untuk melakukan *line balancing* menggunakan metode sekuensial.
Tujuannya adalah mengelompokkan elemen kerja ke dalam beberapa stasiun kerja dengan tetap memperhatikan waktu siklus (*cycle time*).

1 Kode Program

```python
waktu_elemen = [45, 30, 20, 55, 25, 40, 35, 20]
nama_elemen  = ["A", "B", "C", "D", "F", "F", "G", "H"]
cycle_time = 100

stasiun = 1
waktu_stasiun = 0
print(f"Stasiun {stasiun}:"):
for i in range(len(waktu_elemen)):
   if waktu_stasiun + waktu_elemen[i] <= cycle_time:
      print(f"  {nama_elemen[i] ((waktu_elemen[i]):")
      waktu_stasiun += waktu_elemen[i]
   else:
      stasiun += 1.
      waktu_stasiun = waktu_elemen[i]
      print(f"\nStasiun {stasiun}:")
      print(f"  {nama_elemen[i] ((waktu_elemen[i]):")
```

2 Hasil Output

```
Stasiun 1:
  A (45)
  B (30)
  C (20)

Stasiun 2:
  D (55)
  E (25)

Stasiun 3:
  F (40)
  G (35)
  H (20)
```

Metode sekuensial menempatkan elemen kerja secara berurutan hingga mendekati atau mencapai waktu siklus, kemudian dilanjutkan ke stasiun berikutnya.

Tujuan

Menyeimbangkan beban kerja antar stasiun agar proses produksi lebih efisien dan tidak terjadi waktu menganggur yang besar.



Variabel yang Digunakan

- *waktu_elemen* : waktu masing-masing elemen kerja (detik).
- *nama_elemen* : nama elemen kerja.
- *cycle_time* : waktu siklus setiap stasiun (detik).
- *stasiun* : nomor stasiun kerja.
- *waktu_stasiun* : akumulasi waktu pada stasiun saat ini.

Manfaat

- Membantu merancang tata letak lini produksi.
- Mengurangi waktu menganggur pada stasiun kerja.
- Meningkatkan efisiensi dan produktivitas.
- Dapat dikembangkan untuk metode *line balancing* lainnya.

18

Studi Kasus: Monitoring Produktivitas Lini Perakitan

Struktur Kontrol Perulangan

Pada *studi kasus* ini, kita akan membuat program untuk memonitor produktivitas lini perakitan selama beberapa shift dan beberapa hari menggunakan perulangan. Program akan menghitung total unit yang diproduksi, rata-rata per shift, dan mengidentifikasi shift dengan produktivitas tertinggi.

1 Kode Program

```python
data_produksi = [
    [120, 135, 128],   # Hari 1 (shift 1, 2, 3)
    [142, 138, 145],   # Hari 2
    [130, 128, 136],   # Hari 3
    [148, 152, 149],   # Hari 4
    [155, 150, 158]    # hari 5
]
total_hari = len(data_produksi)
total_shift = len(data_produksi[0])
total_unit = 0
shift_tertinggi = 0
produktivitas_shift = [0] * total_shift.

for i in range(total_hari):
    for j in range(total_shift):
        total_unit += data_produksi[i][j]
        produktivitas_shift[j] += data_produksi[i][j]

rata_rata_shift = [p/total_hari for p in produktivitas_shift]
    shift_tertinggi = produktivitas_shift.index(kam(produktivitas_shift)) + 1

    print("Total unit produksi :; total_unit)
    print("Rata-rata per shift "; rata_rata_shift)
    print("Shift dengan produktivitas tertinggi "; shift_tertinggi)
```

2 Hasil Output

```
Total unit produksi : 2191
Rata-rata per shift : [139.0, 140.6, 143.2]
Shift dengan produktivitas tertinggi : 3

Data Produksi (unit)
---------------------------------------
Hari  |  Shift 1  |  Shift 2  |  Shift 3
---------------------------------------
  1   |    120    |    135    |    128
  2   |    142    |    138    |    145
  3   |    130    |    128    |    136
  4   |    148    |    152    |    149
  5   |    155    |    150    |    158
---------------------------------------
```

Tujuan

Memantau dan menganalisis produktivitas lini perakitan untuk mendukung pengambilan keputusan peningkatan efisiensi produksi.



Poin Penting

- Menggunakan *nested loop* untuk mengolah data dua dimensi (hari × shift).
- Menghitung total, rata-rata, dan mencari nilai maksimum.
- Contoh penerapan perulangan pada masalah nyata di industri.

Manfaat

- Mempermudah monitoring dan evaluasi produksi.
- Membantu mengidentifikasi shift paling produktif.
- Dapat dikembangkan untuk analisis data produksi yang lebih kompleks.
- Relevan untuk dunia industri dan manufaktur.

19

Pembahasan Studi Kasus

Struktur Kontrol Perulangan

Pada bagian ini, kita membahas hasil *studi kasus monitoring* produktivitas lini perakitan, menjelaskan alur program, ıгogram, interpretasi hasil output, serta hal-hal penting yang perlu diperhatikan dalam penerapannya.

1 Alur Program

1. **Input Data Produksi**
   Data jumlah unit yang dihaslikan untuk setiap hari dan shift dimasukkan dalam bentuk list dua dimensi.
2. **Inisialisasi Variabel**
   Menghitung total hari, total shift, dan menyiapkan variabel untuk mexyimpan hasil perhitungan.
3. **Perulangan Bersarang (Nested Loop)**
   Melakukan perulangan pada setiap hari dan shift untuk manjumlahkan total unit produksi.
4. **Perhitungan Rata-rata**
   Menghitung rata-rata produksi per shift dan mencari shift dengan produktivitas tertinggi.
5. **Menampilkan Hasil**
   Output berupa total unit produksi, rata-rata per shift, dan informasi shift dengan produktivitas tertinggi.

2 Analisis Hasil Output

Hari	Shift 1	Shift 2	Shift 3	Total per Hari
1	120	135	128	383
2	142	138	145	425
3	130	128	136	394
4	148	152	149	449
5	155	150	158	463
Total	**695**	**703**	**716**	**2.191**

Ringkasan Hasil:

- Total unit produksi selama 5 hari: 2.191 unit
- Rata-rata produksi per shift: 139,0 unit
- Shift dengan produktivitas tertinggi: Shift 3 (rata-rata 143,2 unit)

3 Pembahasan

- Terjadi peninghatan produktivitas dari hari 1 hingga hari 5, yang menunjukkan kinerja lini perakitan semakin baik.
- Shift 3 memiliki rata-rata produksi tertinggi dibandingkan shift lainnya.
- Perulangan memudahkan pengolahan data dalam jumlah besar secara otomatis dan efisien.
- Struktur perulangan dapat dikembangkan lebih lanjut untuk menganalisis faktor lain, seperti waktu henti mesin, tingkat cacat produk, atau efisiensi operator.

Data yang diolah dengan baik, menghasilkan keputusan yang lebih baik!



Kesimpulan

- Studi kasus ini menunjukkan bahwa struktur kontrol perulangan sangat efektif untuk
- mengolah data produksi secara sistematis.
- Informasi yang dihasilkan dapat digunakan sebagai dasar pengambilan keputusan untuk meningkatkan produktivitas lini perakitan.

Saran Implementasi

- Terapkan program serupa pada data aktual di industri.
- Tambahkan analisis lanjutan seperti visualisasi grafik.
- Dapat dikembangkan menjadi sistem monitoring real-time.
- Gunakan data yang lebih lengkap untuk menghasilkan *insight* yang lebih mendalam.

Pertemuan 10
Struktur Kontrol Perulangan

Terima Kasih

Sampai Jumpa di Pertemuan Selanjutnya

> *"Teruslah belajar, karena setiap baris kode adalah langkah kecil menuju solusi yang lebih besar."*
>
> — Belajar Hari Ini, Membangun Masa Depan

Terus Belajar	Terus Menganalisis	Terus Memecahkan Masalah	Terus Berkembang



See You Next Time!

Struktur Data Koleksi

List, Tuple, Set, dan Dictionary untuk Pengelolaan Data Industri

[ ]	( )	{ }	{k:v}
List	**Tuple**	**Set**	**Dictionary**
antrean job	lokasi rak	pemasok unik	data mesin



Data Terstruktur Industri Lebih Efisien



Capaian Pembelajaran Pertemuan Ini

Pertemuan 11 · Struktur Data Koleksi

Learn
Analyze
Solve
Grow



SUB-CPMK

Mahasiswa mampu memilih dan mengolah struktur data koleksi yang tepat untuk menyimpan dan memanipulasi data industri.

Level kognitif: **C3 – C4** (Menerapkan, Menganalisis)

Indikator Ketercapaian

01	Menjelaskan perbedaan karakteristik **list**, **tuple**, **set**, dan dictionary.
02	Melakukan operasi tambah, ubah, hapus, urut, dan cari data pada koleksi.
03	Menulis **list comprehension** untuk transformasi dan penyaringan data.
04	Menyusun **nested dictionary** untuk data terstruktur seperti **Bill of Materials**.

Tujuan Akhir Sesi

Target sesi ini **mencapai level C4:** mahasiswa tidak hanya bisa memakai setiap struktur data, tetapi juga menganalisis struktur mana yang paling tepat untuk suatu kasus.

Penilaian

- Kuis tertulis singkat di akhir sesi
- Tugas penyusunan Bill of Materials dalam nested dictionary dan identifikasi mesin unik dengan set

DARI DATA MENUJU SOLUSI INDUSTRI NYATA

Peta Materi

Pertemuan 11 · Struktur Data Koleksi

"Dari struktur data menuju solusi nyata di dunia industri."



1 Konsep Koleksi

Empat struktur data dan karakteristiknya

Slide 4

→

2 List

[ ]

Indeks, slicing, ubah data, sorting, comprehension

Slide 5 – 8

→

3 Tuple

( )

Data tetap dan unpacking

Slide 9

→

4 Set

{ }

Data unik dan operasi himpunan

Slide 10 – 11

→

5 Dictionary

{k:v}

Key-value, iterasi, nested dict, memilih struktur

Slide 12 – 15

→

6 Promax & Studi Kasus

BOM explosion, aturan penjadwalan, gudang suku cadang

Slide 16 – 20

"Struktur data yang tepat membuat analisis dan keputusan industri menjadi lebih cepat, akurat, dan efisien."

Saran Waktu

55 menit konsep | 30 menit promax dan studi kasus | 15 menit kuis

Fokus

List mendapat porsi terbesar.
Dictionary menjadi fondasi menuju Pandas di pertemuan 14.

Learn
Analyze
Solve
Grow

Learn
Analyze
Solve
Grow

Empat Struktur Data Koleksi Python

Karakteristik dan Contoh Penggunaan di Industri

Jenis	Simbol	Berurutan (Ordered)	Bisa diubah (Mutable)	Duplikat	Contoh di industri
List	`[ "J1", "J2" ]`	✓ Ya	✓ Ya	✓ Boleh	Antrean job, data output harian
Tuple	`("A", 3, 2)`	✓ Ya	✗ Tidak	✗ Boleh	Lokasi rak, batas spesifikasi
Set	`{"PT A", "PT B"}`	✗ Tidak	✓ Ya*	✗ Tidak	Daftar pemasok unik, kompetensi
Dictionary	`{"kode": "M1"}`	✓ Ya**	✓ Ya	✗ Kunci unik unik	Data mesin, stok per kode part

Pertanyaan Kunci saat Memilih

1. Apakah urutan penting?
2. Apakah data boleh berubah?
3. Apakah harus unik?
4. Apakah setiep data perlu diberi nama (kunci)?



Keterangan

* Elemen bisa ditambah/dihapus, tetapi tidak bisa diubah langsung lewat indeks.
** Urutan penyisipan terjaga sejak Python 3.7.

Inti Pemahaman

Masing-masing struktur data memiliki karakter dan kegunaan yang berbeda. Pilih struktur yang paling sesuai dengan kebutuhan data industri agar program lebih **efisien, aman**, dan **mudah dirawat.**

List: Indeks dan Slicing

Mengakses, mengambil sebagian, dan mengecek data dalam list

Learn
Analyze
Solve
Grow

```python
antrean = ["J101", "J102", "J103", "J104", "J105"]
print(antrean[0])        # J101 (pertama)
print(antrean[-1])       # J105 (terakhir)
print(antrean[1:4])      # ['J102', 'J103', 'J104']
print(len(antrean))      # 5
print("J103" in antrean) # True
```

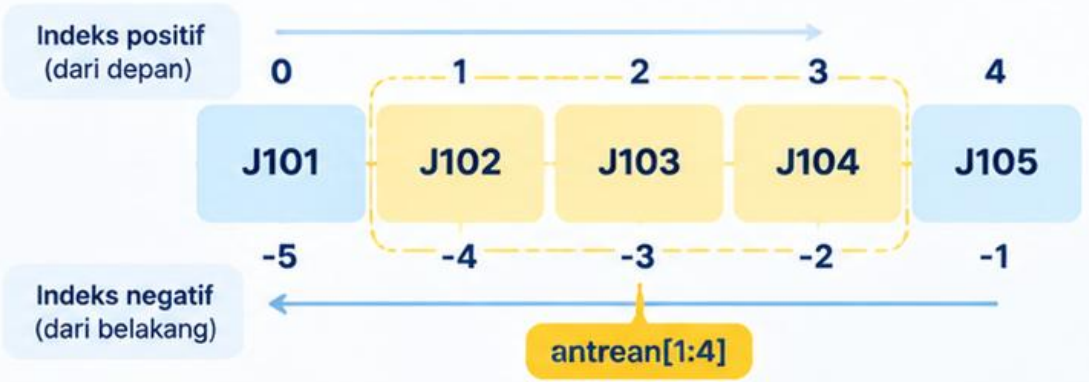
Penjelasan

- Slicing **[awal:akhir]** mengambil sebagian list.
- Seperti **range()**, indeks akhir tidak ikut.
- **[1:4]** berarti indeks 1, 2, 3.
- Indeks dimulai dari 0.
- Mengakses **antrean[5]** akan menghasiikan IndexError.



"Data yang terstruktur membuat keputusan industri semakin akurat."

Peta Indeks Antrean



Contoh Kasus di Industri

List sering digunakan untuk menyimpan antrean job di sebuah work center, data output harian, atau daftar pesanan produksi yang diproses secara berurutan.



Catatan

Mengakses indeks di luar rentang, misalnya **antrean[5]**, akan menghasiikan error: **IndexError**.

Operasi Dasar pada List

Menambah, mengubah, menghapus, mengurutkan, dan mencari data

Learn
Analyze
Solve
Grow

"List membanta kita mengelola data yang dinamis dan selalu berkembang di dunia industri."



Operasi	Contoh Kode	Hasil / Keterangan
Membuat list	`antrean = ["J101", "J102"]`	List dengan 2 elemen
Menambah elemen	`antrean.append("J103")`	`['J101', 'J102', 'J103']`
Menyisipkan elemen	`antrean.insert(1, "J200")`	`['J101', 'J200', 'J102', 'J103']`
Mengubah elemen	`antrean[0] = "J150"`	`['J150', 'J200', 'J102', 'J103']`
Menghapus elemen	`antrean.remove("J102")`	`['J150', 'J200', 'J103']`
Urutkan list	`antrean.sort()`	`['J103', 'J150', 'J200']`
Urutkan terbalik	`antrean.sort(reverse=True)`	`['J200', 'J150', 'J103']`
Cari elemen	`"J150" in antrean`	True
Jumlah elemen	`len(antrean)`	3

Contoh Implementasi

Python

```python
# Antrean job di work center
antrean = ["J101", "J102", "J103"]
antrean.append("J104")
antrean.insert(1, "J200")
antrean.remove("J102")
antrean.sort()
print(antrean)
# Output: ['J101', 'J103', 'J104', 'J200']
```

Aplikasi di Industri

- Antrean job produksi
- Daftar inspeksi harian
- Daftar pesanan pelanggan

Tips

- Gunakan **append()** untuk menambah di akhir.
- Gunakan **insert(t)** jika perlu menyisipkan di posisi tertentu.
- Gunakan **sort(:)** untuk mengurutkan data seperti kode job atau tanggal.

Catatan

Jika mengakses indeks yang tidak ada, Python akan menghasiikan **IndexError**.

List Comprehension

Cara cepat untuk membuat dan memfilter list

"Kode yang singkat, hasil yang lebih kuat untuk data industri."



Bentuk Dasar

Membuat list baru dari iterable

```python
# Kuadrat 1 sampai 5
kuadrat = [x**2 for x in range(1, 6)]
print(kuadrat)
# Output: [1, 4, 9, 16, 25]
```

Dengan Kondisi

Menyaring data berdasarkan kondisi

```python
# Job dengan durasi >= 4 jam
durasi = [2, 5, 3, 8, 4]
job_berat = [d for d in durasi if d >= 4]
print(job_berat)
# Output: [5, 8, 4]
```

Dengan Transformasi

Mengolah data saat membuat list

```python
# Ubah ke huruf besar
part = ["m1", "m2", "m3"]
part_upper = [p.upper() for p in part]
print(part_upper)
# Output: ["M1", "M2", "M3"]
```

Contoh Penerapan di Industri

Filter Job Prioritas

```
job = ["J101", "J102", "J103", "J104"]
prioritas = [j for j in job if j.startswith('
("J10")
# ["J101", "J102", "J103", "J104"]
```

Memilih job berdasarkan kode atau pola tertentu.

Konversi Satuan Produksi

```
kg = [10, 25, 40]
ton = [x/1000 for x in kg]
# [0.01, 0.025, '0.04]
```

Mengubah data satuan secara cepat dan konsisten.

Ekstraksi Data dari Dictionary

```
mesin = [
  ("kode":"M1","jenis":"CNC"],
  ["kode":"M2","jenis":"LATHE") ]
kode_mesin = [m["kode"] for m in mesin]
# ["M1", "M2"]
```

Mengambil informasi tertentu dari data terstruktur.

Keunggulan

- Kode lebih singkat dan mudah dibaca
- Cocok untuk data yang besar
- Mengurangi penggunaan loop manual
- Banyak digunakan dalam analisis data

Catatan

List comprehension tetap menghasliikan list baru di memori.

Metode List yang Sering Digunakan

Fungsi bawaan untuk memanipulasi data dengan lebih praktis

"Metode list membuat pengolahan data industri jadi lebih efisien."

python

Learn
Analyze
Solve
Grow

MinMetode	Fungsi	Contoh Kode	Hasil / Keterangan
append()	Menambah elemen di akhir list	`job.append("J106")`	['J101', 'J102', ..., 'J106"]
insert()	Menyisipkan elemen di posisi tertentu	`job.insert(2, "J200")`	Elemen 'J200' di indeks 2
remove()	Menghapus elemen berdasarkan nilai	`job.remove("J102")`	Elemen 'J102' terhapus
pop()	Menghapus dan mengembalikan elemen	`last = job.pop()`	Menghapus elemen terakhir
sort()	Mengurutkan list (naik)	`job.sort()`	List terurut naik (in-place)
sort(reverse=True)	Mengurutkan list (turun)	`job.sort(reverse=True)`	List terurut turun
index()	Mencari indeks elemen	`job.index("J103")`	Mengembalikan indeks (misal 1)
count()	Menghitung jumlah elemen tertentu	`job.count("J101")`	Jumlah kemunculan (misal 1)
clear()	Menghapus semua elemen	`job.clear()`	List menjadi kosong

Contoh Implementasi

Python

```python
# Daftar job produksi
job = ["J101", "J102", "J103"]
job.append("J104")
job.insert(1, "J200")
job.remove("J101")
job.sort()
print(job)
# Output: ["J102", "J103", "J104", "J200"]
```



Tips Penggunaan

- Gunakan **append()** untuk penambahan cepat.
- Gunakan **insert()** jika posisi elemen penting.
- Gunakan **sort()** untuk mengurutkan data seperti prioritas job atau tanggal.
- Gunakan **index()** dan **count()** untuk pencarian sederhana.
- Metode list sangat berguna dalam penjadwalan, manajemen stok, dan analisis data produksi.

Catatan

sort() mengubah list asli **(in-place)**. Jika ingin menjaga list asli tetap utama, gunakan **sorted(job)** yang menghasilkan list baru.

Tuple: Data Tetap dan Unpacking

Untuk data yang tidak berubah dan memiliki struktur tetap

> *"Tuple menjaga data penting tetap konsisten dan aman dari perubahan yang tidak disengaja."*

python

Learn
Analyze
Solve
Grow



Apa itu Tuple?

Tuple adalah struktur data koleksi yang berurutan, **tidak dapat diubah (immutable)**, dan biasanya digunakan untuk menyimpan data yang bersifat tetap.

```python
# Contoh tuple di industri: lokasi rak (zona, lorong, tingkat)
lokasi = ("A", 3, 2)
print(lokasi)          # ('A', 3, 2)
print(lokasi[0])       # A
print(lokasi[1])       # 3

# Unpacking tuple
zona, lorong, tingkat = lokasi
print(zona)            # A
print(lorong)          # 3
print(tingkat)         # 2
```

Karakteristik Tuple

Berurutan
Elemen memiliki indeks.

Tidak dapat diubah
Setelah dibuat, isi tidak bisa diubah.

Tidak memperbolehkan duplikat
Secara konsep lebih aman untuk data tetap.

Ditulis dengan tanda kurung ( )
Contoh: ("A", 3, 2)

Contoh Kasus di Industri



Tuple sering digunakan untuk menyimpan lokasi rak di gudang, batas spesifikasi mesin, atau koordinat posisi yang tidak berubah.

Contoh Data Lokasi Rak

```
lokasi_rak = ("A", 3, 2)
             ("A")  3)  2)
               ↓    ↓    ↓
             Zona Lorong Tingkat
```

Keunggulan Tuple

- Lebih aman karena tidak bisa diubah.
- Cocok untuk data tetap (konstanta).
- Memori lebih efisien dibanding list.
- Dapat digunakan sebagsi key dalam dictionary.

Catatan

Jika mencoba mengubah elemen tuple, Python akan menghasilkan error: **TypeError**.

Set: Data Unik dan Operasi Himpunan

Menyimpan elemen unik serta melakukan operasi himpunan untuk analisis data industri

"Set membanta kita menghilangkan duplikat dan menemukan informasi unik dengan cepat."

python

Learn
Analyze
Solve
Grow



Apa itu Set?

Set adalah struktur data koleksi yang menyimpan **elemen unik (tidak ada duplikat)**, **tidak berurutan**, bersifat **mutable** (bisa ditambah atau dihapus), dan sangat berguna untuk operasi himpunan.

Contoh Pembuatan Set

Python

```python
# Set otomatis menghapus duplikat
pemasok = {"PT A", "PT B", "PT A", "PT C"}
print(pemasok)    # {'PT A', "PT B", "PT C"}

# Membuat set dari list
mesin = set(["M1", "M2", "M2", "M3"])
print(mesin)      # {"M1", "M2", "M3"}
```

Operasi Dasar pada Set

Operasi	Contoh Kode	Hasil / Keterangan
Menambah elemen	`pemasok.add("PT D")`	Menambah satu elemen
Menghapus elemen	`pemasok.remove("PT B")`	Menghapus elemen (error jika tidak ada)
Menghapus aman	`pemasok.discard("PT X")`	Tidak error jika tidak ada
Jumlah elemen	`len(pemasok)`	Mengembalikan jumlah elemen
Cek keanggotaan	`"PT A" in pemasok`	True / False
Menggabungkan	`set1 | set2`	Union (gabungan)
Irisan	`set1 & set2`	Intersection (irisan)
Selisih	`set1 - set2`	Difference (selisih)
Elemen unik	`set1 ^ set2`	Symmetric difference (pembeda)

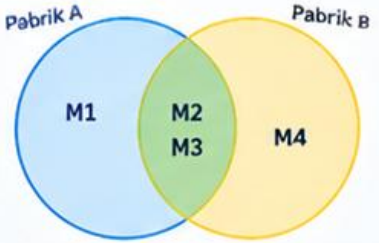
Contoh Operasi Himpunan

Python

```python
pabrik_A = {"M1", "M2", "M3"}
pabrik_B = {"M2", "M3", "M4"}

print(pabrik_A | pabrik_B)  # {'M1', "M2", "M3", "M4"}
print(pabrik_A & pabrik_B)  # {"M2", "M3"}
print(pabrik_A - pabrik_B)  # {"M1"}
print(pabrik_B - pabrik_A)  # {"M4"}
print(pabrik_A ^ pabrik_B)  # {"M1", "M4"}
```

Ilustrasi Himpunan



Penerapan di Industri

- Mengidentifikasi pemasok unik.
- Menentukan mesin unik dari beberapa lini produksi.
- Menghapus data duplikat pada hasil inspeksi.
- Analisis perbedaan komponen antar pabrik.

Keunggulan Set

- Otomatis menghilangkan duplikat.
- Operasi himpunan mudah dan cepat.
- Cocok untuk analisis data industri.
- Lebih efisien dibanding list untuk cek keanggotaan.

Catatan

Set tidak memiliki indeks, sehingga tidak bisa diakses dengan posisi seperti **set[0]**.

"Data membantu kita mengambil keputusan yang lebih tepat dalam pengelolaan SDM."

Learn
Analyze
Solve
Grow

Operasi Himpunan: Mencocokkan Kompetensi Operator

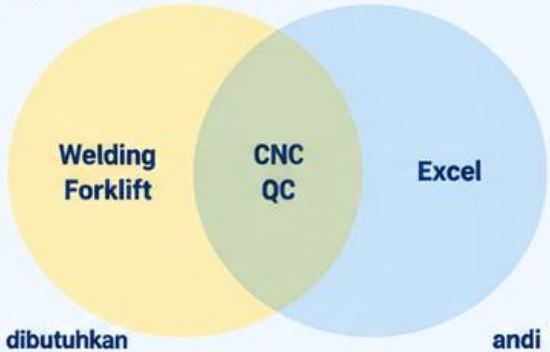
Studi kasus: Membandingkan kebutuhan kompetensi dengan kompetensi yang dimiliki operator

Python

```python
# Data kebutuhan kompetensi dan kompetensi Andi
dibutuhkan = {"CNC", "Welding", "QC", "Forklift"}
andi = {"CNC", "QC", "Excel"}

# Operasi himpunan
print(dibutuhkan & andi)    # intersection
print(dibutuhkan - andi)    # difference
print(dibutuhkan | andi)    # union
print(dibutuhkan ^ andi)    # symmetric difference
```

Skill Matrix sebagai Diagram Venn



- Kebutuhan pelatihan Andi
- Kompetensi tambahan Andi
- Kompetensi sesuai (sudah dimiliki)

Penjelasan Operasi Himpunan

& Intersection (irisan)
Menunjukkan kompetensi yang sesuai antara kebutuhan dan yang dimiliki.

- Difference (selisih)
Menunjukkan kompetensi yang masih dibutuhkan (perlu pelatihan).

| Union (gabungan)
Menggabungkan seluruh kompetensi dari kedua himpunan.

^ Symmetric difference
Menunjukkan kompetensi yang berbeda dari kedua himpunan (tidak sama).

Hasil Operasi Himpunan

Operasi	Hasil	Keterangan
& (intersection)	{'CNC', 'QC'}	Kompetensi yang sesuai
- (difference)	{'Welding', 'Forklift'}	Kebutuhan pelatihan Andi
| (union)	5 kompetensi	Gabungan seluruh kompetensi
^ (symmetric difference)	{'Excel', 'Forklift', 'Welding'}	Kompetensi yang berbeda

Implementasi di Industri



Operasi himpunan dapat digunakan dalam manajemen SDM operator produksi untuk mengidentifikasi gap kompetensi, menentukan kebutuhan pelatihan, dan menyusun program pengembangan keterampilan secara lebih objektif dan berbasis data.

Dictionary: Pasangan Kunci dan Nilai

Menyimpan data dalam bentuk pasangan kunci (key) dan nilai (value)

"Data yang terstruktur memudahkan kita mengelola informasi di dunia industri."

python

Learn
Analyze
Solve
Grow



Apa itu Dictionary?

Dictionary adalah struktur data yang menyimpan **pasangan kunci dan nilai**, di mana setiap **kunci harus unik** dan digunakan untuk **mengakses** nilai yang terkait.

Contoh Dictionary (Data Mesin)

Python

```python
# Data mesin industri
mesin = {
    "kode": "CNC-01",
    "jam_operasi": 1200,
    "status": "Aktif"
}
print(mesin)  # {'kode': 'CNC-01', 'jam_operasi': 1200, 'status': 'Aktif'}
```

Contoh Data Dictionary

Kunci (Key)	Nilai (Value)	Keterangan
kode	'CNC-01'	Kode mesin
jam_operasi	1200	Total jam operasi
status	'Aktif'	Status mesin
lokasi	'Line A'	Lokasi mesin
maintenance	False	Status perawatan

Operasi Dasar Dictionary

Operasi	Contoh Kode	Hasil / Keterangan
Mengakses nilai	`mesin["kode"]`	'CNC-01'
Akses aman	`mesin.get("daya", 0)`	0 (jika tidak ada)
Mengubah nilai	`mesin["status"] = "Idle"`	Nilai diperbarui
Menambah key baru	`mesin["tahun"] = 2024`	Key baru ditambahkan
Menghapus key	`mesin.pop("lokasi")`	Menghapus key 'lokasi'
Melihat semua key	`mesin.keys()`	`dict_keys([...])`
Melihat semua nilai	`mesin.values()`	`dict_values([...])`

Kapan Digunakan?

- Menyimpan data dengan label/kategori.
- Mengelola informasi yang kompleks.
- Mudah diakses menggunakan kunci.
- Cocok untuk data industri (mesin, operator, produk, dll).

Tips

- Gunakan nama key yang jelas dan konsisten.
- Manfaatkan `get()` untuk menghindari error.
- Dictionary bisa berisi tipe data apa saja.

Menelusuri Dictionary: keys(), values(), items()

Studi kasus: Laporan stok suku cadang dan pembuatan Purchase Order (PO) otomatis

"Data yang terkelola **dengan baik mendukung efisiensi operasional** industri."

python | Learn Analyze Solve Grow

Kode Program

Python

```python
stok = {"BR-101": 35, "BR-205": 8, "SH-310": 50, "OR-022": 4}
MIN = 10

print(list(stok.keys()))      # semua kode part
print(sum(stok.values()))     # total unit

for kode, qty in stok.items():
    if qty < MIN:
        print(f"{kode}: {qty} unit -> buat PO")
```

Output Program

```
['BR-101', 'BR-205', 'SH-310', 'OR-022']
07
BR-205: 8 unit -> buat PO
OR-022: 4 unit -> buat PO
```

Penjelasan Metode

keys()	Mengembalikan semua kunci dalam dictionary.
values()	Mengembalikan semua nilai dalam dictionary.
items()	Mengembalikan pasangan (kunci, nilai) sebagai tuple.

Contoh Data Stok Suku Cadang

Kode Part	Jumlah (unit)
BR-101	35
BR-205	8
SH-310	50
OR-022	4

Stok di bawah MIN (perlu PO)

Catatan Penting

- **for kode in stok** hanya menelusuri kunci.
- Gunakan **items()** saat membutuhkan kunci dan nilai sekaligus.
- Cocok digunakan untuk membuat laporan stok dan otomatisasi pembuatan Purchase Order (PO) pada data yang memenuhi kondisi tertentu.

Implementasi di Industri

- Memantau stok suku cadang secara real-time.
- Mengidentifikasi item yang perlu dipesan ulang (PO) otomatis.
- Dapat dikembangkan untuk integrasi dengan sistem persediaan (*inventory system*).

Nested Dictionary: Data Berlapis

Struktur data bertingkat untuk menyimpan informasi mesin industri.

"Data yang terstruktur membantu kita mengelola sistem industri dengan lebih efektif dan efisien."

python
Learn
Analyze
Solve
Grow



Kode Program

Python

```python
armada = {
    "CNC-01": {"lokasi": "Line A", "jam": 1250},
    "CNC-02": {"lokasi": "Line B", "jam": 480},
}
print(armada["CNC-02"]["lokasi"])    # Line B

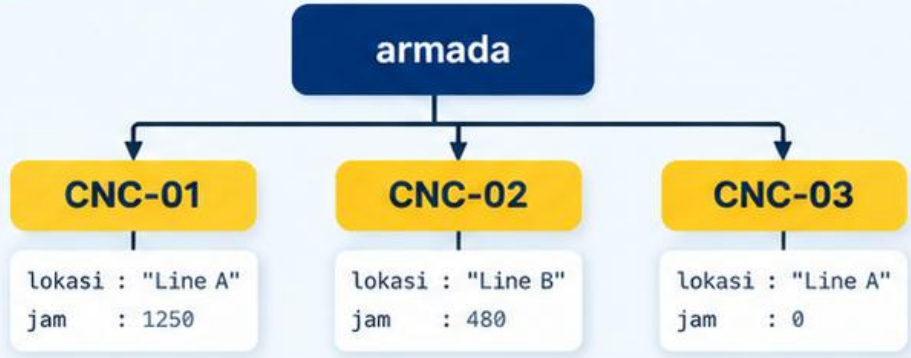
armada["CNC-03"] = {"lokasi": "Line A", "jam": 0}

for kode, info in armada.items():
    status = "PM" if info["jam"] >= 1000 else "OK"
    print(kode, info["lokasi"], status)
```

Output Program

```
CNC-01 Line A PM
CNC-02 Line B OK
CNC-03 Line A OK
```

Struktur Pohon Data



Akses Berlapis

```
armada["CNC-02"]["lokasi"]  →  "Line B"
```

Catatan Penting

Nilai dalam dictionary dapat berupa dictionary lain sehingga membentuk data berlapis seperti pohon.

Aplikasi di Industri

- Data armada mesin produksi
- Penentuan jadwal *preventive maintenance*
- Monitoring jam operasi mesin

Memilih Struktur Data yang Tepat

Gunakan karakteristik data dan kebutuban operasi untuk menentukan struktur data yang paling sesuai.

"Struktur data yang tepat membuat program lebih efisien, mudah dipelihara, dan siap dikembangkan."



1. Perlu mengakses data dengan **nama/kode**? → **dict**
2. Harus **unik** atau butuh **operasi himpunan**? → **set**
3. Data **tetap** dan **tidak boleh berubah**? → **tuple**
4. **Urutan** penting dan data **sering berubah**? → **tist**

*Periksa pertanyaan dari atas; ambil jawaban **Ya pertama**.*

POLA PALING BERGUNA

List of dictionary: satu dict = satu baris data.

```python
produksi = [
    {"tgl": "01-11", "unit": 520},
    {"tgl": "02-11", "unit": 480},
]
```

Bentuk ini identik dengan tabel dan akan langsung menjadi **DataFrame Pandas** di **Pertemuan 14.**



"Dari struktur produk ke perencanaan kebutuhan material yang lebih akurat."

"Plan
Schedule
Control
Optimise

MRP

Contoh 1: BOM Explosion (Dasar MRP)

Menggunakan nested dictionary untuk merepresentasikan struktur produk dan menghitung kebutuhan bahan.

Kode Program (Python)

Python

```python
bom = {"Meja": {"Kaki": 4, "Papan": 1, "Baut": 8},
       "Kaki": {"Pipa": 1, "Karet": 1, "Baut": 2}}
stok = {"Kaki": 40, "Papan": 20, "Baut": 300,
        "Pipa": 50, "Karet": 500}
pesanan = 50                                  # meja

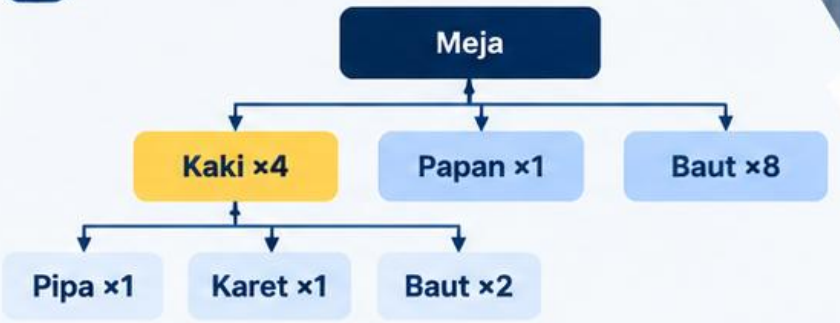
kotor = {}
for komp, qty in bom["Meja"].items():         # level 1
    kotor[komp] = kotor.get(komp, 0) + qty * pesanan
buat_kaki = max(kotor.pop("Kaki") - stok["Kaki"], 0)
for sub, qty in bom["Kaki"].items():          # level 2
    kotor[sub] = kotor.get(sub, 0) + qty * buat_kaki

print(f"Produksi kaki: {buat_kaki} unit")
for bahan, q in kotor.items():
    beli = max(q - stok[bahan], 0)
    print(f"{bahan:6}: butuh {q:4}, stok {stok[bahan]:4}, beli {beli}")
```

Catatan

Nested dictionary digunakan untuk merepresentasikan struktur produk (BOM) dengan beborapa level.

Struktur Produk (BOM)



Kaki = Pipa ×1 + Karet ×1 + Baut ×2

Hasil Perhitungan Kebutuhan Bahan

Bahan	Butuh	Stok	Beli
Papan	50	20	30
Baut	720	300	420
Pipa	160	50	110
Karet	160	500	0

Produksi kaki: 200 - 40 = 160 unit

Contoh: Membandingkan Aturan Penjadwalan

Membandingkan aturan penjadwalan berdasarkan rata-rata flow time dan jumlah keterlambatan.

"Pemilihan aturan penjadwalan yang tepat dapat meningkatkan efisiensi produksi."



Kode Program (Python)

```python
jobs = [("J1", 6, 10), ("J2", 2, 6), ("J3", 8, 18),
        ("J4", 3, 15), ("J5", 4, 12)]   # (nama, proses, due)

def evaluasi(urutan):
    t, total_flow, telat = 0, 0, 0
    for nama, proses, due in urutan:
        t += proses                 # waktu selesai job
        total_flow += t
        if t > due:
            telat += 1
    return total_flow / len(urutan), telat

def proses(j): return j[1]
def due(j): return j[2]

aturan = {"FCFS": jobs,
          "SPT": sorted(jobs, key=proses),
          "EDD": sorted(jobs, key=due)}

for nama, urutan in aturan.items():
    rata, telat = evaluasi(urutan)
    print(f"{nama:4}: flow {rata:.1f, terlambat {telat}")
```

Hasil Perbandingan Aturan Penjadwalan

Aturan	Urutan	Flow	Telat
FCFS	1–2–3–4–5	14.4	3
SPT	2–4–5–1–3	10.8	2
EDD	2–1–5–4–3	12.0	1

Rata-rata Flow Time (jam)



Kesimpulan

- SPT unggul di flow time.
- EDD unggul di ketepatan waktu.
- Tidak ada aturan terbaik untuk semua tujuan.

Insight Industri

Pemilihan aturan penjadwalan harus disesuaikan dengan tujuan produksi, apakah meminimalkan flow time atau meminimalkan keterlambatan.

"Data suku cadang yang terkelola dengan baik menjaga keandalan mesin dan kelancaran produksi."

Inventory for a Reliable Industry

Studi Kasus: Gudang Suku Cadang Maintenance

PT Andalan Mekanika (fiktif) menyimpan data suku cadang dalam nested dictionary: kode part sebagai kunci, atribut sebagai dictionary, dan lokasi rak sebagai tuple (zona, baris, level).



Data Suku Cadang

Kode	Nama	Stok	Min	Harga (Rp)	Pemasok	Rak
BR-101	Bearing 6205	35	20	45.000	PT Rolindo	(A, 1, 2)
BR-205	Bearing 6305	8	15	62.000	PT Rolindo	(A, 1, 3)
VB-040	V-Belt B40	12	10	38.000	CV Sabuk	(B, 2, 1)
OR-022	O-Ring 22mm	40	50	2.500	CV Seal	(C, 4, 1)
FL-310	Filter Oli	5	6	85.000	PT Filtra	(B, 3, 2)
SH-120	Shaft 20mm	9	4	150.000	PT Rolindo	(D, 1, 1)

Laporan yang diminta kepala gudang

1. Daftar part di bawah stok minimum (list comprehension)
2. Pemasok yang harus dihubungi, tanpa duplikat (set)
3. Total nilai persediaan (stok × harga)
4. Lokasi rak part FL-310 (tuple unpacking) dan jumlah PO per part

Tugas Kelompok (15 menit)

Susun data ke **nested dictionary**, lalu buat keempat laporan. Jumlah PO = 2 × stok minimum − stok saat ini.

Gunakan struktur data yang tepat untuk mengelola data gudang secara efisien.

Pembahasan Studi Kasus

Analisis hasil pengolahan data gudang suku cadang menggunakan nested dictionary

"Data yang terkelola dengan baik mendukung kelancaran operasional dan efisiensi industri."



Kode Program (Python)

Python

```python
gudang = {
    "BR-205": {"nama": "Bearing 6305", "stok": 8, "min": 15,
               "harga": 6200, "pemasok": "PT Rolindo",
               "rak": ("A", 1, 3)},
    # ... 5 part lainnya dengan struktur yang sama
}

kritis = [k for k, p in gudang.items() if p["stok"] < p["min"]]
hubungi = {gudang[k]["pemasok"] for k in kritis}    # set

nilai = 0
for p in gudang.values():
    nilai += p["stok"] * p["harga"]

zona, baris, level = gudang["FL-310"]["rak"]
# proses selanjutuya untuk membuat PO
```

Hasil Laporan

Part kritis	BR-205, OR-022, FL-310
Pemasok	PT Rolindo, CV Seal, PT Filtra
Nilai persediaan	Rp4.402.000
Lokasi FL-310	Zona B, baris 3, level 2

Output

```
PO BR-205: 22 unit ke PT Rolindo
PO OR-022: 60 unit ke CV Seal
PO FL-310: 7 unit ke PT Filtra
```

Konsep yang digunakan

- **List comprehension** (mencari part kritis)
- **Dictionary** (struktur data utama)
- **Set** (menghindari duplikasi pemasok)
- **Tuple unpacking** (lokasi rak)

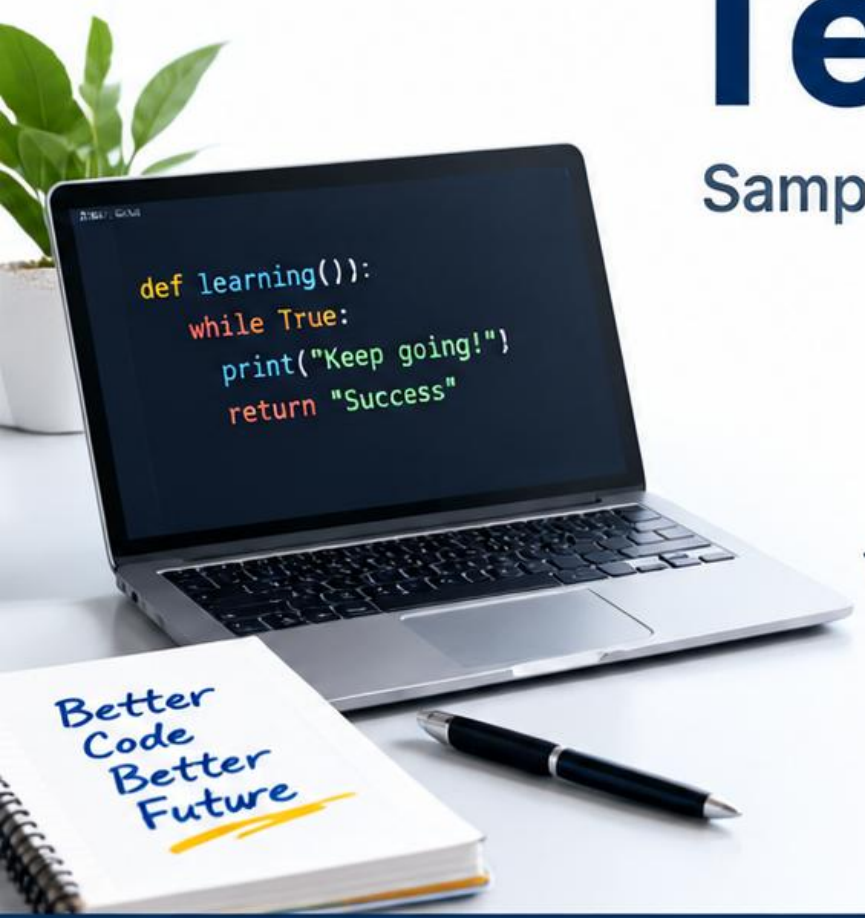
Diskusi

Jika BR-101 ikut kritis, berapa kali PT Rolindo muncul di set?

"Konsistensi dalam belajar adalah kunci untuk mencapai hasil yang luar biasa."

Terima Kasih

Sampai jumpa di pertemuan berikutnya!



Terus Belajar

Terus Berkarya

Raih Masa Depan



PERTEMUAN 12

Modul, Pustaka Standar, dan Error Handling

Membuat program industri yang cerdas memakai alat siap pakai dan tangguh terhadap kesalahan

import | math | random | datetime | pip | try-except

"Kode yang baik tidak hanya bekerja, tetapi juga siap menghadapi kesalahan."

Better Data Better Industry



Dari Kode untuk Industri yang Lebih Baik

Capaian Pembelajaran

Pertemuan 12 · Modul, Pustaka Standar, dan Error Handling

Better Data Better Industry



SUB-CPMK

Mahasiswa mampu memanfaatkan **modul bawaan Python** dan membuat program yang **tahan terhadap kesalahan input** pada permasalahan industri.

Level kognitif: C3 (Menerapkan)

> *"Program yang baik bukan hanya menghasilkan output yang benar, tetapi juga tetap berjalan meski menghadapi kesalahan."*

Indikator Ketercapaian

1. Mengimpor dan memakai modul **math, statistics, random**, dan **datetime**, serta membuat modul sendiri.
2. Memasang library eksternal menggunakan **PIP**.
3. Menangani error dengan **try, except, else, finally**, dan **raise**.
4. Menyajikan laporan rapi dengan **f-string** dan **memvalidasi input** pengguna.

Peta Materi

Pertemuan 12 · Modul, Pustaka Standar, dan Error Handling

"Small Modules Big Impact"

Better Data Better Industry



1 — Konsep Modul
Analogi kotak perkakas, cara import, modul buatan sendiri
Slide 4 – 6

2 — Pustaka Standar
math, statistics, random, datetime
Slide 7 – 11

3 — PIP
Memasang library dari PyPI
Slide 12

4 — Mengenal Error
Jenis kesalahan, exception umum, membaca traceback
Slide 13 – 14

5 — Menangani Error
try-except, else, finally, raise, validasi input
Slide 15 – 19

6 — Laporan & Promax
f-string, studi waktu, Monte Carlo, studi kasus
Slide 20

Fokus Pertemuan 12

Kita akan memanfaatkan modul yang sudah ada dan membuat program yang tahan terhadap kesalahan input pada permasalahan industri.

Saran Alokasi Waktu

- Modul & pustaka standar : ± 35 menit
- Error handling : ± 35 menit
- Promax & studi kasus : ± 30 menit

Apa Itu Modul?

Pertemuan 12 · Modul, Pustaka Standar, dan Error Handling

"Jangan menulis ulang yang sudah ada."

Modul adalah file Python berisi kumpulan fungsi dan variabel yang **siap dipakai** program lain.

Analogi: teknisi maintenance tidak membuat kunci pas sendiri setiap kali memperbaiki mesin. Ia mengambil alat dari kotak perkakas.
Modul adalah kotak perkakas bagi programmer.

Intinya

Jangan menulis ulang yang sudah ada. Gunakan modul agar kode lebih singkat, teruji, dan minim kesalahan.

Sumber Modul

Pustaka standar

Sudah terpasang bersama Python.
Contoh: math, statistics, random, datetime.

Library pihak ketiga

Dipasang dari PyPI dengan pip.
Contoh: numpy, pandas, matptotlib.

Modul buatan sendiri

File .py milik kita, misalnya kumpulan rumus Teknik Industri.



Empat Cara Mengimpor Modul

Pertemuan 12 · Modul, Pustaka Standar, dan Error Handling

"Import the tools, not the problems."



No.	Cara	Pemakaian	Kapan dipakai
1	`import math`	`math.sqrt(16)`	Standar, paling jelas asal fungsinya
2	`import statistics as st`	`st.mean(data)`	Nama modul panjang, pakai alias
3	`from math import sqrt, ceil`	`sqrt(16)`	Hanya butuh beberapa fungsi
4	`from math import *`	`sqrt(16)`	Hindari: nama bisa bertabrakan

Contoh Penggunaan

```
import math
print(math.pi)          # 3.141592653589793
print(dir(math)[:5])    # intip isi modul
help(math.ceil)         # baca dokumentasi
```

Output (contoh)

```
3.141592653589793
['__doc__', '__file__', '__loader__',
 '__name__', '__package__']
Help on built-in function ceil in module math:
ceil(x, /)
    Return the ceiling of x as an integer.
```

Intinya

Import di baris paling atas program. Gunakan **dir()** untuk melihat isi modul dan **help()** untuk membaca penjelasan fungsi.

Gunakan cara import yang paling sesuai dengan kebutuhan program Anda.

Membuat Modul Sendiri

Pertemuan 12 · Modul, Pustaka Standar, dan Error Handling

"Modul hari ini, solusi untuk masalah esok."

Better Data Better Industry

1 Buat file modul (.py)

rumus_ti.py

```python
# rumus_ti.py
# Modul berisi rumus sederhana Teknik Industri

def eoq(D, S, H):
    """Menghitung Economic Order Quantity (EOQ)"""
    return (2 * D * S / H) ** 0.5

def oee(availability, performance, quality):
    """Menghitung Overall Equipment Effectiveness"""
    return availability * performance * quality
```

2 Gunakan modul di program lain

main.py

```python
# main.py
from rumus_ti import eoq, oee

D = 1200     # permintaan per tahun
S = 500      # biaya pesan per order
H = 2000     # biaya simpan per unit per tahun

hasil = eoq(D, S, H)
print(f"EOQ = {hasil:.2f} unit")

nilai_oee = oee(0.90, 0.85, 0.95)
print(f"OEE = {nilai_oee:.2%}")
```

3 Struktur file proyek

```
proyek_ti/
    rumus_ti.py  # modul buatan sendiri
    main.py      # program utama
```

Hasil Output

```
> python main.py
EOQ = 24.49 unit
OEE = 72.68%
```

Manfaat Modul Sendiri

- Kode lebih rapi dan terstruktur
- Bisa dipakai ulang di proyek lain
- Mudah dibagikan dengan tim
- Mengurangi kesalahan karena rumus sudah diuji

Intinya

Modul sendiri membantu kita menyimpan rumus yang sering digunakan, sehingga kode lebih singkat, rapi, dan mudah dipelihara.



Modul math

Pertemuan 12 · Modul, Pustaka Standar, dan Error Handling

"Rumus besar dimulai dari fungsi sederhana."

Modul **math** menyediakan fungsi **matematika dasar dan lanjutan** yang sudah teruji dan siap digunakan.

Kapan digunakan?

- **Perhitungan matematis** (akar, log, trigonometri)
- **Perhitungan numerik** di bidang Teknik Industri
- Lebih akurat dan efisien daripada menulis rumus sendiri

Contoh Penggunaan

Python

```python
import math
print(math.pi)              # 3.141592653589793
print(math.sqrt(16))        # 4.0
print(math.ceil(4.2))       # 5
print(math.floor(4.8))      # 4
print(math.log(10))         # 2.302585092994046
print(math.sin(math.pi/2)) # 1.0
print(math.radians(180))    # 3.141592653589793
```

Fungsi yang Sering Digunakan

Fungsi	Kegunaan	Contoh	Hasil
`math.pi`	Nilai π	`math.pi`	3.14159
`math.e`	Bilangan e	`math.e`	2.71828
`math.sqrt(x)`	Akar kuadrat	`math.sqrt(16)`	4.0
`math.ceil(x)`	Pembulatan ke atas	`math.ceil(4.2)`	5
`math.floor(x)`	Pembulatan ke bawah	`math.floor(4.8)`	4
`math.log(x)`	Logaritma natural	`math.log(10)`	2.3026
`math.sin(x)`	Sinus (radian)	`math.sin(math.pi/2)`	1.0
`math.cos(x)`	Cosinus (radian)	`math.cos(0)`	1.0
`math.tan(x)`	Tangen (radian)	`math.tan(math.pi/4)`	1.0
`math.radians(x)`	Derajat ke radian	`math.radians(180)`	3.14159
`math.degrees(x)`	Radian ke derajat	`math.degrees(math.pi)`	180.0

Intinya

Gunakan modul **math** untuk perhitungan matematika yang sudah tersedia, agar kode lebih singkat, akurat, dan minim kesalahan.



Modul random

Pertemuan 12 · Modul, Pustaka Standar, dan Error Handling

"Keacahan hari ini, bisa jadi solusi di industri besok."

Better Data Better Industry

Modul **random** digunakan untuk menghasilkan **angka atau data secara acak** yang berguna dalam simulasi, pemodelan, atau pengujian program.

Contoh Penggunaan

Python

```python
import random
# 1. random float
print(random.random())
# 2. random integer
print(random.randint(1, 10))
# 3. random pilihan
buah = ['apel', 'jeruk', 'mangga', 'pisang']
print(random.choice(bauh))
# 4. ambil 2 sampel acak
print(random.sample(baah, 2))
```

Fungsi yang Sering Digunakan

Fungsi	Kegunaan
`random()`	Menghasilkan float antara 0.0 – 1.0
`randint(a, b)`	Menghasilkan bilangan bulat a – b
`uniform(a, b)`	Menghasilkan float antara a – b
`choice(seq)`	Memilih satu elemen secara acak
`sample(seq, k)`	Memilih k elemen acak tanpa pengulangan
`shuffle(seq)`	Mengacak urutan elemen (in-place)

Output (contoh)

```
0.6394267984578837   # hasil random()
7                    # hasil randint(1,10)
mangga               # hasil choice()
['jeruk', 'pisang']  # hasil sample()
```

Intinya

Gunakan modul **random** untuk kebutuhan data acak.

- Sangat berguna untuk simulasi, pengujian, dan studi kasus industri.
- Pilih fungsi yang sesuai dengan kebutuhan.



Contoh Kasus di Industri

Simulasi waktu kedatangan pelanggan, permintaan harian, atau uji coba sistem produksi dapat menggunakan modul random.

Catatan

Untuk hasil acak yang sama setiap kali program dijalankan, gunakan **random.seed(nilai)**.

Modul datetime

Pertemuan 12 · Modul, Pustaka Standar, dan Error Handling

"Waktu yang tepat, solusi yang lebih akurat."

Better Data Better Industry

Modul **datetime** digunakan untuk bekerja dengan **tanggal, waktu, dan durasi waktu** pada program Python.

Contoh Penggunaan

Python

```python
from datetime import datetime, timedelta

# Waktu sekarang
now = datetime.now()
print("Sekarang:", now)

# Tambah 7 hari
next_week = now + timedelta(days=7)
print("7 hari lagi:", next_week)

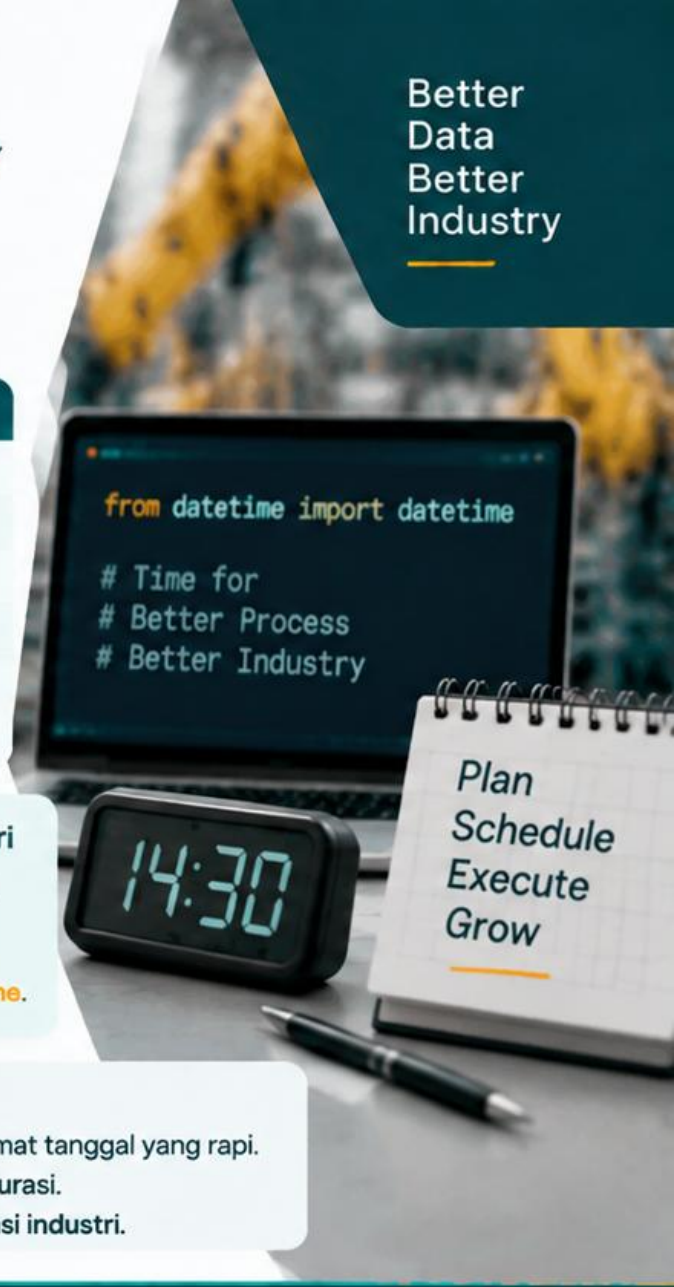
# Format tanggal
print("Format:", now.strfime("%d-%m-%Y %H:%M:%S"))
```

Fungsi yang Sering Digunakan

Fungsi/Kelas	Kegunaan
`datetime.now()`	Mendapatkan tanggal dan waktu saat ini
`datetime.today()`	Sama seperti now()
`datetime(year, m, d)`	Membuat objek tanggal dan waktu
`timedelta(days=..))`	Menambahkan atau mengurangi waktu
`strffime(format)`	Mengubah format tanggal/waktu ke string
`strptime(string, format)`	Mengubah string ke objek datetime

Output (contoh)

```
Sekaroag: 2026-09-25 14:30:12.345678
7 hari lagi: 2026-10-02 14:30:12.345678
Format: 25-09-2026 14:30:12
```



Contoh Kasus di Industri

Pencatatan waktu produksi, penjadwalan maintenance, **atau analisis durasi proses** menggunakan modul **datetime**.

Intinya

Gunakan modul **datetime** untuk mengelola **tanggal dan waktu** dalam program agar proses lebih terjadwal dan terukur.

Tips

- Gunakan strftime() untuk menampilkan format tanggal yang rapi.
- Manfaatkan timedelta untuk perhitungan durasi.
- Sesuaikan format dengan kebutuhan aplikasi industri.

Modul pip

Pertemuan 12 · Modul, Pustaka Standar, dan Error Handling

"Install pengetahuan baru, upgrade kemampuan setiap hari."

Better Data Better Industry

pip adalah **package manager** untuk menginstal, mengelola, dan menghapus pustaka (library) Python dari Python Package Index (PyPi).

Kegunaan pip

- Menginstal pustaka pihak ketiga
- Memperbarui pustaka
- Menghapus pustaka
- Melihat daftar pustaka yang terinstal
- Mengelola versi pustaka

Kapan digunakan?

Digunakan ketika kita membutuhkan pustaka (paket) tambahan yang tidak termasuk dalam pustaka standar Python.

Contoh Penggunaan

Python

```python
# 1. Install pustaka
pip install numpy

# 2. Install versi tertentu
pip install pandas==2.2.0

# 3. Upgrade pustaka
pip install --upgrade matplotlib

# 4. Hapus pustaka
pip uninstall seaborn

# 5. Lihat daftar pustaka yang terinstal
pip list

# 6. Lihat informasi pustaka
pip show numpy
```

Contoh Output (pip list)

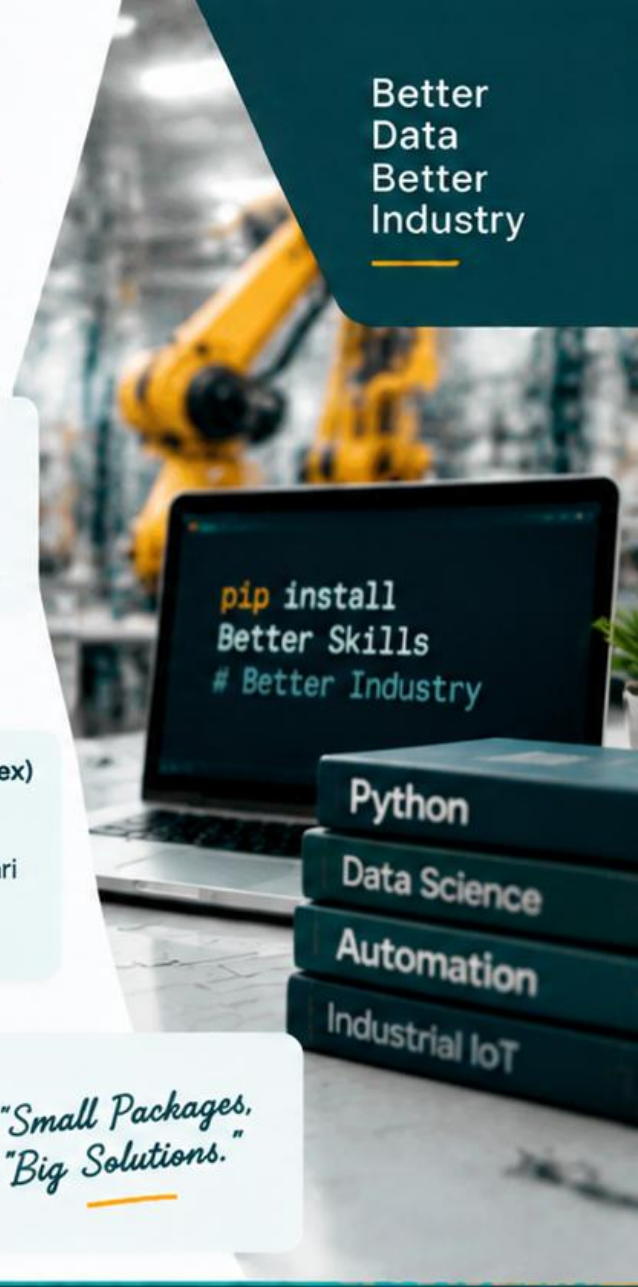
```
Package          Version
-----------------------------------
numpy            1.26.4
pandas           2.2.0
matplotlib       3.8.2
seaborn          0.13.2
pip              24.0
setuptools       69.0.3
```

Sumber Pustaka

PyPI (Python Package Index)

https://pypi.org

Tempat resmi untuk mencari dan mengunduh ribuan pustaka Python.



Tips

- Pastikan koneksi internet stabil saat menginstal.
- Gunakan virtual environment (venv) untuk setiap proyek.
- Periksa dokumentasi pustaka sebelum digunakan.
- Gunakan pip freeze > requirements.txt untuk menyimpan daftar pustaka.

"Small Packages, Big Solutions."

Modul try-except

Pertemuan 12 · Modul, Pustaka Standar, dan Error Handling

"Error bukan akhir, tapi awal untuk program yang lebih tangguh."

Better Data Better Industry

try-except digunakan untuk **menangani kesalahan (error)** agar program tidak berhenti secara tiba-tiba dan tetap berjalan dengan aman.

Struktur Dasar try-except

```
try:
    # kode yang berpotensi error
except TipeError:
    # penanganan jika terjadi error
```

Contoh 1 - Pembagian dengan try-except

Python

```
try:
    a = 10
    b = 0
    hasil = a / b
    print("Hasil:", hasil)
except ZeroDivisionError:
    print("Tidak bisa membagi dengan nol!")
```

Output

```
Tidak bisa membagi
dengan nol!
```

Penjelasan

Program tidak berhenti meskipun terjadi kesalahan pembagian dengan nol.

Contoh 2 - Input angka dengan try-except

Python

```
try:
    angka = int(input("Masukkan angka: "))
    print("Anda memasukkan:", angka)
except ValueError:
    print("Input bukan angka!n
    Silakan masukkan angka.")
```

Output (contoh)

```
Masukkan angka: abc
Input bukan angka!
Silakan massukkan angka.
```

Penjelasan

Program menangani kesalahan jika input bukan angka.

Jenis Error yang Sering Terjadi

Error	Keterangan
ZeroDivisionError	Pembagian dengan nol
ValueError	Tipe data tidak sesuai
FileNotFoundError	File tidak ditemukan
IndexError	Akses indeks di luar jangkauan
TypeError	Tipe data tidak cocok

Tips

- Gunakan **try-except** untuk menghindari crash pada program.
- Tangani error spesifik (mis. ValueError, FileNotFoundError).
- Berikan pesan kesalahan yang jelas dan informatif.
- Gunakan ***finally*** untuk kode yang selalu diigankan.

Pengembangan Lanjut

Kita juga dapat menggunakan blok **else** dan **finally** untuk mengelola alur program dengan lebih baik.



Kode yang siap menghadapi dunia nyata.

Modul module

Pertemuan 12 · Modul, Pustaka Standar, dan Error Handling

"Kode yang rapi hari ini, proyek yang lebih besar esok hari."

Better Data Better Industry

Module adalah file Python (.py) yang berisi fungsi, variabel, dan kode yang dapat **digunakan kembali** di program lain.

Keuntungan Menggunakan Module

- Kode lebih rapi dan terstruktur
- Dapat digunakan kembali (reusable)
- Memudahkan kolaborasi
- Memisahkan fungsi berdasarkan tugas

Contoh 1 – Membuat dan Menggunakan Module Sendiri

mymodule.py

```
def salam(nama):
    return f"Halo, {nama}!"

def tambah(a, b):
    return a + b
```

main.py

```
import mymodule
print(mymodule.salam("Teknik Industri"))
print("Hasil tambah:",
      mymodule.tambah(5, 3))
```

Output

```
Halo, Teknik Industri!
Hasil tambah: 8
```

Penjelasan

File **mymodule.py** berisi fungsi yang digunakan di file lain menggunakan perintah `import`.

Contoh 2 – Menggunakan Module Bawaan

main.py

```
import math
r = 7
luas = math.pi * r**2
print("Luas lingkaran:", luas)
```

Output

```
Luas lingkaran: 153.93804002589985
```

Penjelasan

Module `math` sudah tersedia secara standar di Python dan dapat digunakan langsung dengan perintah `import`.

Cara Import Module

- `import module` → mengimpor seluruh module
- `from module import fungsi` → mengimpor fungsi tertentu
- `import module as alias` → memberi nama alias

Contoh Module Bawaan yang Sering Digunakan

Module	Kegunaan
math	Fungsi matematika (pi, sqrt, sin, cos, dll)
random	Bilangan acak
datetime	Tanggal dan waktu
os	Interaksi dengan sistem operasi
sys	Informasi sistem dan argumen program

Tips

- Gunakan module untuk membuat kode lebih rapi dan mudah dipelihara.
- Manfaatkan module bawaan sebelum membuat sendiri.
- Buat module sendiri untuk fungsi yang sering digunakan.



"Modul Kecil, Dampak Besar."

13

Mengenal Error: Tiga Jenis Kesalahan Program

"Pahami error, bangun program yang lebih andal."

Better Data Better Industry

Syntax error

Salah tulis aturan bahasa; program tidak bisa jalan sama sekali.

```python
print("OK"
```

Exception (runtime)

Sintaks benar, tetapi gagal saat dijalankan karena kondisi tertentu.

```python
480 / 0
```

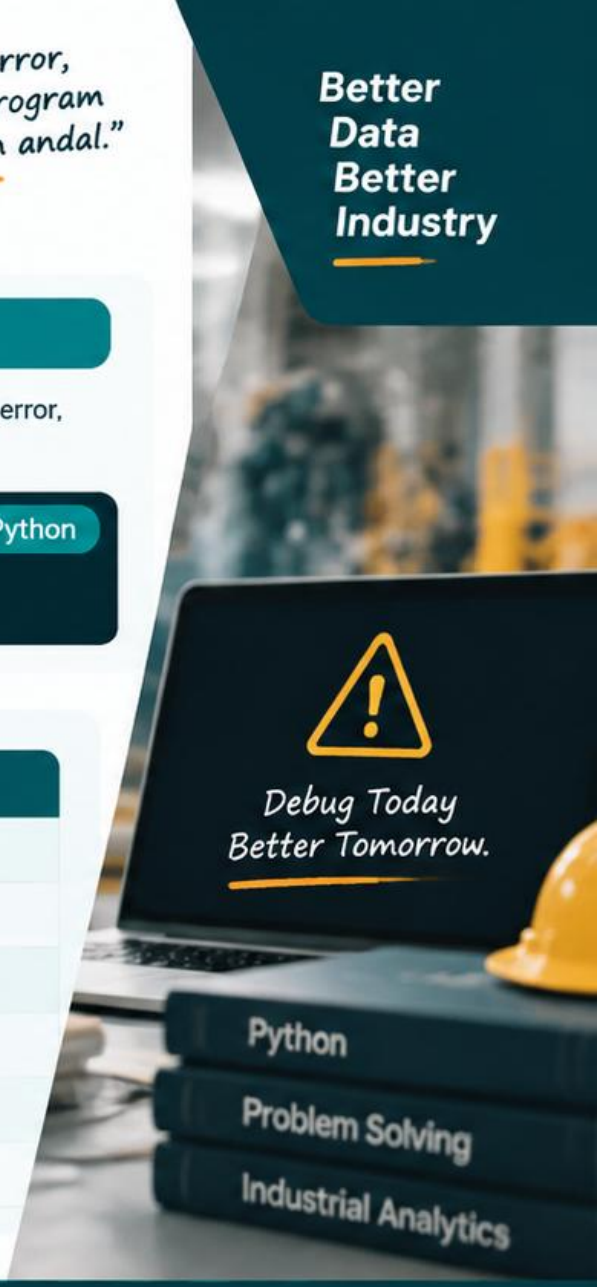
Logic error

Program jalan tanpa pesan error, tetapi hasilnya salah.

```python
urutan elif keliru (P9)
```

Exception

Exception	Contoh pemicu	Situasi di industri
ValueError	int("500")	Operator salah ketik huruf O
ZeroDivisionError	output / 0	Jam kerja diisi 0
KeyError	stok["XX-999"]	Kode part tidak terdaftar
IndexError	antrean[10]	Posisi di luar panjang list
TypeError	"480" + 20	Teks dijumlah dengan angka
FileNotFoundError	open("log.csv")	File tidak ada (P13)



Membaca Traceback:
Pesan Error Adalah Petunjuk

"Setiap pesan error adalah petunjuk untuk menjadi programmer yang lebih baik."

Better Data Better Industry

Python

```
Traceback (most recent call last):
  File "/content/laporan.py", line 8, in <module>
    rate = cacat / output
           ~~~~~~~~~~^~~~~~~~
ZeroDivisionError: division by zero
```

1 Baca baris terakhir dulu

Jenis error dan pesannya: ZeroDivisionError, pembagian dengan nol.

2 Lihat kode penyebab

Tanda ^ menunjuk operasi yang gagal: cacat / output.

3 Cari lokasinya

File laporan.py baris 8. Buka baris itu dan periksa nilai variabelnya.



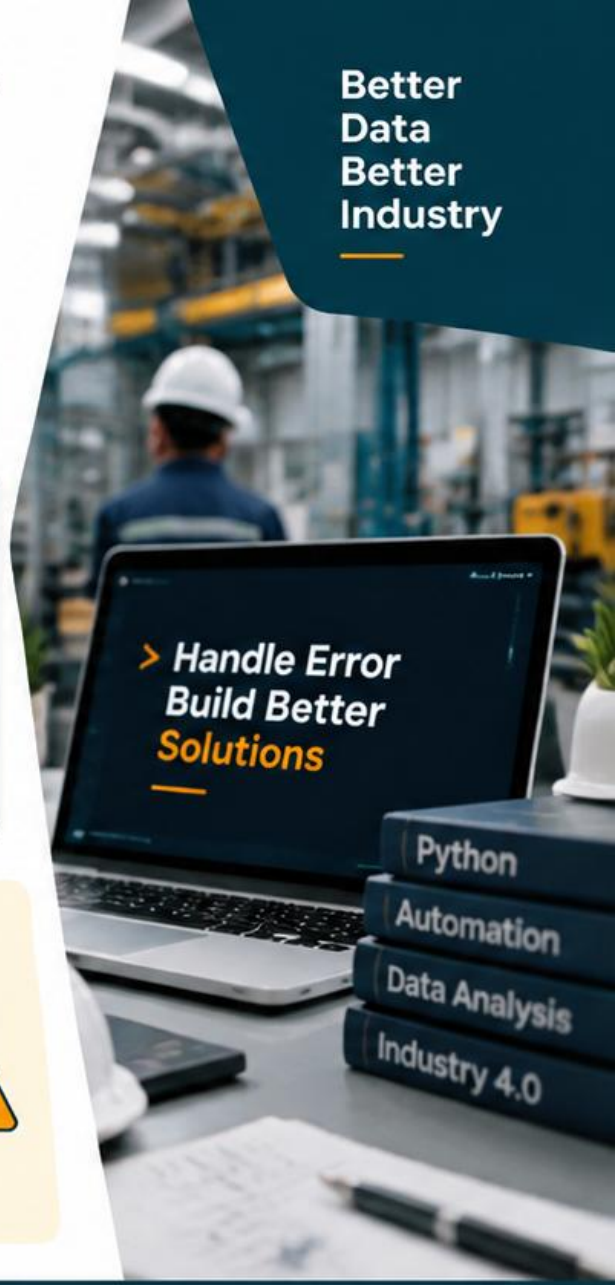
Intinya

Traceback dibaca dari bawah ke atas. Jangan panik melihat pesan merah: isinya memberi tahu apa, di mana, dan kenapa.

Most recent call last = pemanggilan terakhir ada di paling bawah.

try-except: Menangkap Error agar Program Tidak Mogok

"Tangani error hari ini, agar program tetap berjalan esok hari."



TANPA try

Jika terjadi error, program langsung berhenti.

```python
output = input("Output: ")    # user: 480
print(int(output) / 8)
# ValueError -> program BERHENTI
```

ValueError: invalid literal for int() with base 10: 'abc'

DENGAN try-except

Error ditangani, program tetap berjalan.

```python
output = input("Output: ")    # user: 480
try:
    print(int(output) / 8)
except ValueError:
    print("Output harus berupa angka.")
print("Program tetap berjalan")
```

Cara kerja

1. Python mencoba menjalankan blok try.
2. Jika berhasil, blok **except** dilewati.
3. Jika muncul error yang cocok, sisa blok try dibatalkan dan blok **except** dijalankan.
4. Program lanjut ke baris setelahnya.

Intinya

try = "coba lakukan",
except = "kalau gagal, lakukan ini".

Mirip SOP darurat di pabrik.
Saat ada masalah, kita punya prosedur cadangan agar proses tetap berjalan.



Menangani Beberapa Jenis Error Sekaligus

Satu blok try-except dapat menangani beberapa jenis error sesuai kebutuhan.

"Error ditangani, program tetap memberi solusi."

Better Data Better Industry

Kode Program

Python

```python
def produktivitas(output, jam):
    try:
        hasil = int(output) / float(jam)
    except ValueError:
        return "Input harus angka"
    except ZeroDivisionError:
        return "Jam kerja tidak boleh 0"
    return f"{hasil:.1f} unit/jam"

print(produktivitas("480", "8"))
print(produktivitas("480", "8"))
print(produktivitas("480", "0"))
```

Output Program

```
60.0 unit/jam
Input harus angka
Jam kerja tidak boleh 0
```

Menyimpan pesan error: as e

Python

```python
try:
    int("480")
except ValueError as e:
    print("Detail:", e)
```

Hindari except kosong

except: atau except **Exception:** menangkap semua error, termasuk salah ketik nama variabel. Bug tersembunyi dan sulit dilacak. Pakai hanya sebagai jaring pengaman terakhir, dengan mencetak e.

> Tangani error secara spesifik dan informatif.





else dan finally: Melengkapi Alur Penanganan

"Error ditangani, program tetap memberi solusii."

Blok try-except dapat dilengkapi dengan **else dan finally** untuk mengatur alur eksekusi saat berhasil, saat terjadi **error**, maupun pekerjaan yang selalu dijalankan.

Kode Program

Python

```python
try:
    rate = cacat / output
except ZeroDivisionError:
    print("Tidak ada produksi hari ini")
else:
    print(f"Defect rate: {rate:.2%}") # hanya jika sukses
finally:
    print("[log] inspeksi harian ditutup") # selalu
```

Output Program

```
# Jika berhasil (output > 0)
Defect rate: 5.00%
[log] inspeksi harian ditutup

# Jika terjadi error (output = 0)
Tidak ada produksi hari ini
[log] inspeksi harian ditutup
```

Penjelasan Setiap Bagian

try	Kode yang mungkin menghasilkan **error**.
except	Dijalankan jika terjadi **error**. Pada contoh ini menangani **ZeroDivisionEfror**.
else	Dijalankan **hanya jika tidak ada error** (kode pada **try** berhasil).
finally	Selalu dijalankan, baik terjadi **error** maupun tidak. Biasanya digunakan untuk menutup file, menyimpan log, atau membersihkan **resource**.

Gunakan finally untuk proses wajjb

finally cocock digunakan untuk pekerjaan yang harus selalu dijalankan, misalnya:

- Menutup file
- Menyimpan log
- Menutup koneksi database
- Membersiihkan resource





raise: Menegakkan Aturan Bisnis

"Error ditangani, program tetap memberi solusi."

Tidak semua data salah memicu error Python. raise digunakan untuk membuat error secara sengaja ketika aturan bisnis dilanggar.

Kode Program

Python

```python
def defect_rate(output, cacat):
    if output <= 0:
        raise ValueError("Output harus lebih dari 0")
    if cacat < 0 or cacat > output:
        raise ValueError("Jumlah cacat tidak valid")
    return cacat / output

try:
    print(defect_rate(450, 500))
except ValueError as e:
    print("Data ditolak:", e)
```

Output Program

```
Data ditolak: Jumlah cacat tidak valid
```

Aturan bisnis lain

- Jam kerja > 24 per hari
- Tanggal produksi di masa depan
- Suhu di luar rentang sensor

Intinya

raise mengubah pelanggaran aturan menjadi error yang bisa ditangkap except.



Pola Validasi Input yang Tangguh

Kombinasi while, try-except, dan raise menghasilkan validasi input yang lebih aman, fleksibel, dan mudah digunakan.

"Error ditangani, program tetap memberi solusi."

Better Data Better Industry

Kode Program

Python

```python
def minta_angka(pesan, minimum=0):
    while True:
        try:
            nilai = float(input(pesan))
            if nilai < minimum:
                raise ValueError
            return nilai
        except ValueError:
            print(f"Masukkan angka >= {minimum}.")

jam = minta_angka("Jam kerja: ", minimum=1)
output = minta_angka("Output (unit): ")
print(f"Produktivitas: {output / jam:.1f} unit/jam")
```

Contoh Interaksi

```
Jam kerja: delapan
Masukkan angka >= 1.
Jam kerja: 0
Masukkan angka >= 1.
Jam kerja: 8
Output (unit): 480
Produktivitas: 60.0 unit/jam
```

Perbaikan dari Pertemuan 10

- **isdigit()** menolak angka desimal seperti 7.5.
- Kombinasi **while** + **try** + **raise** menerima desimal.
- Menolak teks dan memeriksa batas minimum.
- Lebih fleksibel dan aman untuk berbagai kasus input di dunia nyata.



Terima Kasih!

Sampai jumpa di pertemuan berikutnya.

Terima kasih telah mengikuti materi **Modul & Error Handling** pada mata kuliah **Pemrograman Komputer**.

- **Teruslah berlatih** dan eksplorasi contoh kasus lainnya.
- **Terapkan konsep** error handling dalam proyek nyata.
- **Siapkan diri** untuk materi berikutnya!

"Error ditangani, program tetap memberi solusi."

> "Kesalahan adalah bagian dari proses belajar, dan penanganan error yang baik adalah langkah d menuju program yang lebih andal."



