Belajar Python

Mohammad Febryan Khamim

1 Perbedaan List, Tuple, Set dan Dictionary

1. List

List adalah tipe data yang ditulis berurutan yang berisi lebih dari satu tipe data.

- **Contoh:** `a = ["teks", 7, 2.5, True]`
- **Bersifat *mutable*:** Bisa ditambahkan atau dikurangi.
  - `.append()` → ditambahkan. Contoh: `a.append("satu")`
  - `del` → dihapus. Contoh: `del a[2]`

2. Tuple

Tuple mirip seperti list tapi isinya ***immutable***, tidak dapat diubah nilainya.

3. Set

Set adalah kumpulan item yang bersifat unik dan tak berurutan.

- **Contoh:** `S = {1, 2, "tuple"}`
- Jika terdapat lebih dari satu data yang sama → hanya tersimpan 1.

```
S = {1, 2, 2, 4}
print(S)  # Output: {1, 2, 4}
```

- **Cara menambahkan data:** Harus menggunakan fungsi `.add()`.

```
S = {1, 2, 2, 4}
S.add(0)
print(S)  # Output: {0, 1, 2, 4}
```

4. Dictionary

Dictionary adalah kumpulan pasangan kunci dan nilai (*key : value*) dan tak berurutan.

- **Format:** `d = {"key": "Value"}`
- **Contoh:**

```
d = {"hewan" : "gajah", "angka" : 1}
```

- **Cara mengubah nilai:**

```
d["hewan"] = "unta"
```

Tabel Perbedaan
Berikut ini adalah tabel perbedaan dari List, Tuple, Set, dan Dictionary.

Fitur	List	Tuple	Set	Dictionary
Simbol	`[ ]`	`( )`	`{ }`	`{k:v}`
Sifat	*Mutable*	*Immutable*	*Mutable*	*Mutable*
Urutan	*Ordered*	*Ordered*	*Unordered*	*Unordered*
Duplikasi	Boleh ada data sama	Boleh ada data sama	**Tidak Boleh** (Unik)	Key harus unik, Value boleh sama
Contoh	`[1, 2, 2]`	`(1, 2, 2)`	`{1, 2}`	`{"a": 1}`

2 IF-ELSE Statement

Dalam Python, terdapat cara untuk menjelaskan terkait *conditional statement*, yakni menggunakan IF ELSE. Berikut ini adalah penjelasannya.

1. **Satu Kondisi**

   Bagian ini digunakan untuk dapat menjalankan suatu perintah ketika sebuah syarat terpenuhi, berikut ini contohnya.

```
nilai = 80
if nilai >= 75:
    print("Selamat, Anda Lulus!")
```

2. **Banyak Kondisi**

   Apabila ingin menjalankan suatu perintah yang memiliki beberapa persyaratan, berikut ini contohnya.

```
skor = 85

if skor >= 90:
    print("Predikat: A")
elif skor >= 80:
    print("Predikat: B")
elif skor >= 70:
    print("Predikat: C")
else:
    print("Predikat: D")
```

3. **Operator Perbandingan dan Logika**

   Berikut ini adalah beberapa operator logika yang sering digunakan dalam IF ELSE dalam Python. Kondisi di dalam `if` sering kali menggunakan operator berikut:

   - **Perbandingan:** $==, \neq, >, <, \geq, \leq$
   - **Logika:**
     - `and` : Benar jika kedua kondisi terpenuhi ($A \wedge B$).
     - `or` : Benar jika salah satu kondisi terpenuhi ($A \vee B$).

3 Looping dalam Python

Untuk membuat perulangan atau *looping* pada Python, terdapat beberapa hal utama yang dapat digunakan dalam Python, di antaranya adalah :

1. **For Loop**

   For Loop digunakan untuk melakukan iterasi di antara beberapa barisan, seperti *list*, *tuple*, *string*, atau *range*. Jumlah iterasinya **sudah diketahui** dari awal.

   Listing 1: Contoh For Loop

```
# Iterasi menggunakan range
for i in range(1, 6):
    print(f"Perulangan ke-{i}")

# Iterasi pada list
buah = ["apel", "mangga", "pisang"]
for item in buah:
    print(f"Saya suka {item}")
```

2. **While Loop**

   Untuk **While Loop** digunakan untuk melakukan perulangan selama masih memenuhi kondisi tertentu (atau kondisi tersebut bernilai **True**).

   Listing 2: Contoh While Loop

```
count = 0
while count < 5:
    print(f"Nilai count: {count}")
    count += 1  # Increment
```

3. **Break dan Continue**

   Pernyataan **kontrol perulangan** (*loop control statements*) mengubah eksekusi dari urutan normalnya.

   - **Break**. Pernyataan `break` digunakan untuk menghentikan perulangan secara paksa, bahkan jika kondisi *loop* sebenarnya masih bernilai *True*. Setelah `break` dipanggil, program akan langsung menjalankan baris kode setelah blok perulangan.

     Listing 3: Contoh Break

```
for i in range(1, 10):
    if i == 5:
        break  # Loop berhenti total saat i bernilai 5
    print(i)
# Output: 1, 2, 3, 4
```

   - **Continue**. Pernyataan `continue` digunakan untuk melompati sisa kode di dalam iterasi saat ini dan langsung berpindah ke iterasi berikutnya. Perulangan tidak berhenti, hanya "loncat" ke pengecekan kondisi berikutnya.

Listing 4: Contoh Continue

```
    for i in range(1, 6):
        if i == 3:
            continue  # Angka 3 akan dilewati
        print(i)
    # Output: 1, 2, 4, 5
```

- **Pass**. Pernyataan `pass` adalah pernyataan *null* (kosong). Perbedaannya dengan komentar adalah Python tidak mengabaikan `pass`, tetapi secara sintaksis `pass` digunakan sebagai penampung (*placeholder*) saat kode belum ditulis, tetapi blok tersebut wajib ada secara struktur.

Listing 5: Contoh Pass

```
for i in range(5):
    if i == 2:
        pass  # Tidak melakukan apa-apa
    print(f"Angka: {i}")
```

4. **Nested Loop** Perulangan di dalam perulangan lainnya.

Listing 6: Contoh Nested Loop

```
for i in range(1, 4):
    for j in range(1, 4):
        print(f"i={i}, j={j}")
```

Library dalam Python

8 Januari 2026

1 Library Pandas

Library Pandas merupakan *open source* yang menyediakan beberapa peralatan untuk kebutuhan analisis, manipulasi, dan pembersihan data. Pandas memiliki format data yang disebut DataFrame untuk membentuknya dalam struktur data 2 dimensi atau tabel. Berikut ini adalah beberapa contoh *syntax* yang umum digunakan dalam Pandas.

- **Syntax mengubah *dictionary* jadi tabel**

Listing 7: Dictionary jadi Tabel

```
# Membuat data dictionary
data = {
    'Nama': ['Anna', 'Bob', 'Charlie', 'David', 'Eva'],
    'Umur': [18, 19, 17, 18, 19],
    'Matematika': [85, 90, 78, 92, 88],
    'Bahasa_Inggris': [88, 85, 92, 89, 94],
    'IPA': [90, 87, 84, 88, 91]
}

# Membuat DataFrame dari data
df = pd.DataFrame(data)

# Menampilkan DataFrame
print("DataFrame Awal:")
print(df)
```

- **Membaca dan Menyimpan Data**

```
# Membaca Data
df = pd.read_csv('data.csv')
df_excel = pd.read_excel('data.xlsx')

# Menyimpan dictionary dalam CSV
df.to_csv('output.csv', index=False)
df.to_excel('output.xlsx', index=False)
```

Adanya *syntax* berupa **index** tersebut menunjukkan apakah indeks (urutan baris) dimasukkan ke dalam *file* yang akan disimpan atau tidak.

- **Melihat Struktur Data**

```
df.head()         # 5 baris pertama
df.tail()         # 5 baris terakhir
df.info()         # informasi kolom & tipe data
df.describe()     # statistik deskriptif
df.shape          # jumlah baris dan kolom
```

- **Seleksi Baris dan Kolom**

```
df['Nama']
df[['Nama', 'Umur']]

df.loc[0]           # berdasarkan index label
df.iloc[0]          # berdasarkan posisi

df.loc[0, 'Nama']   # sel tertentu
```

- **Filtering Data**

```
df[df['Umur'] > 20]
df[df['Kota'] == 'Bandung']
df[(df['Umur'] > 20) & (df['Kota'] == 'Bandung')]
```

- **Mengatasi *Missing Value***

```
df.isnull().sum()
df_drop = df.dropna()
df_fill = df.fillna(0)
```

- **Manipulasi Kolom**

```
df['Total'] = df['Nilai1'] + df['Nilai2']
df.rename(columns={'Nilai1':'Nilai_A'}, inplace=True)
df.drop(columns=['Nilai2'], inplace=True)
```

2 Library PyTorch

PyTorch adalah *framework* Machine Learning (ML) *open-source* yang dikembangkan untuk *deep learning*. *Framework* ini sangat populer karena mendukung komputasi berbasis GPU dan dibuat dengan konsep *dynamic computation graph*, di mana grafik komputasi dibentuk secara dinamis saat program dijalankan.

Fitur Utama PyTorch

- **Dynamic Graphs**: Mempermudah eksekusi dan proses *debug*.
- **Automatic Diff. Engine**: Mempermudah komputasi gradien secara otomatis.
- **Mendukung CUDA**: Memungkinkan komputasi pada GPU untuk mempercepat proses pelatihan.

Komponen Utama

Berikut ini terdapat beberapa komponen yang dapat digunakan pada PyTorch, di antaranya sebagai berikut.

1. **Tensor**

   Tensor merupakan struktur data inti di PyTorch yang serupa dengan *array* pada NumPy, tetapi memiliki kemampuan akselerasi pada GPU. Dalam *deep learning*, Tensor digunakan untuk menyimpan data *input*, *output*, dan parameter model. Tensor digunakan untuk mendukung diferensiasi otomatis yang cocok untuk tugas *deep learning*.

```
import torch
t = torch.tensor([[1, 2, 3, 4],
                  [5, 6, 7, 8],
                  [9, 10, 11, 12]])

print("Reshaping")
print(t.reshape(6, 2))

print("\nResizing")
print(t.view(2, 6))

print("\nTransposing")
print(t.transpose(0, 1))
```

2. **Autograd dan Computational Graphs**

   Autograd secara otomatis dapat digunakan untuk menghitung turunan atau gradien dari suatu operasi yang sangat berguna dalam *backpropagation* pada Neural Network. Sementara itu, Computational Graph merupakan representasi grafis dari operasi matematis yang dilakukan pada tensor.

```
import torch
import torch.nn as nn


class NeuralNetwork(nn.Module):
    def __init__(self):
        super(NeuralNetwork, self).__init__()
        self.fc1 = nn.Linear(10, 16)
        self.fc2 = nn.Linear(16, 8)
        self.fc3 = nn.Linear(8, 1)

    def forward(self, x):
        x = torch.relu(self.fc1(x))
        x = torch.relu(self.fc2(x))
        x = torch.sigmoid(self.fc3(x))
        return x


model = NeuralNetwork()
print(model)
```

3. **Loss Function dan Optimizer**

   - **Loss Function**: Digunakan untuk menghitung tingkat kesalahan (*error*) antara prediksi model dengan label asli.
   - **Optimizer**: Bertugas memperbarui bobot model berdasarkan perhitungan gradien untuk meminimalkan *loss*.

4. **Proses Pelatihan Model**

   Terdapat empat tahapan utama dalam pelatihan model di PyTorch.

   (a) **Forward Pass**: Mengalirkan data melalui model untuk mendapatkan prediksi.

(b) **Perhitungan Loss**: Mengukur seberapa jauh hasil prediksi dari target.

(c) **Backward Pass**: Menghitung gradien *loss* terhadap parameter model.

(d) **Pembaruan Parameter**: Menggunakan *optimizer* untuk memperbarui bobot.

5. **Model Deep Learning pada *pytorch***

   Beberapa model populer yang didukung antara lain.

   - **Convolutional Neural Network (CNN)**: Fokus pada pemrosesan data gambar.
   - **Recurrent Neural Network (RNN)**: Fokus pada data urutan (*sequence*) seperti teks.
   - **Model Generative**: Seperti GAN (*Generative Adversarial Networks*) dan VAE (*Variational Autoencoders*).

6. **Contoh Implementasi Kode**

   Berikut adalah contoh pembuatan Tensor dan operasi dasar di PyTorch.

Listing 8: Operasi Dasar Tensor PyTorch

```
import torch

# Membuat tensor dari list
data = [[1, 2], [3, 4]]
x_data = torch.tensor(data)

# Membuat tensor acak
shape = (2, 3,)
rand_tensor = torch.rand(shape)

# Operasi matematika sederhana
x = torch.tensor([5.0], requires_grad=True)
y = x**2
y.backward() # Menghitung gradien

print(f"Tensor Data: \n {x_data}")
print(f"Gradien dari x^2 pada x=5: {x.grad}") # Hasilnya 2*x =
    10
```

Berikut adalah contoh kode Python untuk mendefinisikan model, menghitung *loss*, dan melakukan pembaruan bobot (optimasi).

Listing 9: Neural Network Sederhana dengan PyTorch

```
import torch
import torch.nn as nn
import torch.optim as optim

# 1. Menyiapkan data dummy (Input & Target)
# Misal: 10 data dengan 5 fitur masing-masing
input_data = torch.randn(10, 5)
target = torch.randn(10, 1)

# 2. Mendefinisikan Arsitektur Model
```

```
class SimpleNet(nn.Module):
    def __init__(self):
        super(SimpleNet, self).__init__()
        # Layer input ke hidden (5 fitur ke 10 neuron)
        self.hidden = nn.Linear(5, 10)
        # Fungsi aktivasi
        self.relu = nn.ReLU()
        # Layer hidden ke output (10 neuron ke 1 output)
        self.output = nn.Linear(10, 1)

    def forward(self, x):
        x = self.hidden(x)
        x = self.relu(x)
        x = self.output(x)
        return x

# 3. Inisialisasi Model, Loss Function, dan Optimizer
model = SimpleNet()
criterion = nn.MSELoss() # Mean Squared Error
optimizer = optim.SGD(model.parameters(), lr=0.01)

# 4. Proses Pelatihan Singkat (5 Epoch)
for epoch in range(5):
    # Forward pass
    prediction = model(input_data)
    loss = criterion(prediction, target)

    # Backward pass & Optimization
    optimizer.zero_grad() # Reset gradien
    loss.backward()       # Backpropagation
    optimizer.step()      # Update bobot

    print(f"Epoch {epoch+1}, Loss: {loss.item():.4f}")
```

Penjelasan Komponen

- `nn.Module`: Kelas dasar untuk semua modul jaringan saraf di PyTorch.
- `forward()`: Menentukan bagaimana data mengalir melalui lapisan-lapisan *neural network*.
- `optimizer.zero_grad()`: Sangat penting untuk membersihkan gradien lama sebelum menghitung gradien baru pada setiap iterasi agar tidak terakumulasi.
- `loss.backward()`: Langkah di mana PyTorch menghitung gradien untuk setiap parameter yang memiliki `requires_grad=True`.

3 Library TensorFlow

TensorFlow adalah sebuah **framework open-source** untuk *machine learning* dan *artificial intelligence* yang dikembangkan oleh Google Brain. *Framework* ini dibuat untuk membangun dan melatih model ML dan *deep learning*.

Dalam penggunaan sehari-hari dengan Python, TensorFlow menyediakan ekosistem yang lengkap termasuk:

- API tingkat rendah untuk mendefinisikan dan menjalankan komputasi numerik.
- Integrasi dengan **Keras**, yaitu API tingkat tinggi untuk membangun model neural network secara intuitif.
- Dukungan alat visualisasi dan debugging seperti TensorBoard.
- Mekanisme optimasi model dan *deployment* untuk perangkat *mobile* dan *web* (mis. TensorFlow Lite dan TensorFlow.js).

TensorFlow memungkinkan penggunaan pada berbagai perangkat komputasi seperti CPU, GPU, dan TPU untuk mempercepat proses pelatihan model.

Kelebihan TensorFlow di antaranya sebagai berikut.

- **Skalabilitas tinggi** untuk riset dan produksi.
- **Ekosistem lengkap** termasuk deployment, model pretrained, dan alat bantu lain
- Mendukung komputasi terdistribusi dan akselerator perangkat keras.
- Ketersediaan **API dalam Python** yang kuat dan fleksibel.

Perbandingan Framework ML/DL

Berikut ini adalah tabel perbandingan antara empat *framework / library* populer dalam *domain machine learning* dan *deep learning*:

Fitur	**TensorFlow**	**PyTorch**	**Keras**	**Scikit-Learn**
Jenis Library	Deep Learning	Deep Learning	High-level DL API	Machine Learning klasik
Tingkat Abstraksi	Rendah–tinggi	Rendah–menengah	Tinggi	Tinggi
Kemudahan Penggunaan	Cukup kompleks	Lebih natural (Pythonic)	Sangat mudah	Sangat mudah
Graf Komputasi	Static + Eager Execution	Dynamic (define-by-run)	Menggunakan backend TensorFlow	Tidak menggunakan computational graph
Fokus Utama	Produksi dan deployment skala besar	Riset dan prototyping	Pembangunan model cepat	Algoritma ML tradisional
Dukungan Deployment	Sangat kuat (TF Lite, TF Serving, TF.js)	Terbatas dibanding TF	Mengikuti TensorFlow	Tidak berfokus pada deployment
Contoh Use Case	CNN, RNN, NLP, Vision	Riset DL, GAN, NLP	Prototyping neural network	Klasifikasi, regresi, clustering

Catatan:

- Keras pada awalnya adalah API tinggi yang berjalan di atas TensorFlow, sehingga memudahkan pembangunan model DL tanpa perlu detail komputasi tingkat rendah.
- Scikit-Learn fokus pada algoritma *machine learning* klasik seperti regresi, SVM, dan pohon keputusan.
- PyTorch dikenal dengan grafik komputasi dinamis yang cocok untuk riset dan *prototyping*.

Arsitektur TensorFlow secara umum dapat terdiri dari beberapa komponen utama sebagai berikut.

- **Tensor**

  Tensor adalah struktur data utama di TensorFlow. Tensor merupakan suatu vektor atau matriks multidimensi yang digunakan untuk menyimpan *input*, parameter, dan *output* model.

- **Computational Graph**

  TensorFlow membangun *graph komputasi* yang terdiri dari ***node*** yakni operasi matematis (penjumlahan, perkalian, aktivasi) serta ***edge*** berupa aliran tensor di antara operasi.

- **Execution**

  TensorFlow memiliki dua model eksekusi yakni Graph Execution (komputasi dioptimalkan terlebih dahulu) dan Eager Execution (hasil langsung dihitung).

Ekosistem TensorFlow mendukung seluruh proses dari pembangunan dan pelatihan model. TensorFlow memiliki beberapa ekosistem di antaranya sebagai berikut.

3.1 TensorFlow Core

API tingkat rendah untuk:

- operasi tensor
- pembuatan computational graph
- kontrol eksekusi

3.2 Keras

API tingkat tinggi di dalam TensorFlow untuk:

- membangun neural network secara sederhana
- menyusun layer secara berurutan
- training dan evaluasi model

3.3 TensorBoard

Alat visualisasi untuk:

- memantau proses training
- melihat grafik loss dan akurasi
- melihat struktur model dan graph

3.4 TensorFlow Lite

Digunakan untuk:

- deployment pada perangkat mobile
- IoT dan edge device
- optimasi model menjadi lebih ringan

3.5 TensorFlow Serving

Digunakan untuk:

- deployment model pada server
- layanan API untuk model machine learning

3.6 TensorFlow.js

Digunakan untuk menjalankan model:

- langsung di browser
- menggunakan JavaScript

3.7 TensorFlow Hub

Repositori yang menyediakan:

- model siap pakai (pre-trained)
- transfer learning