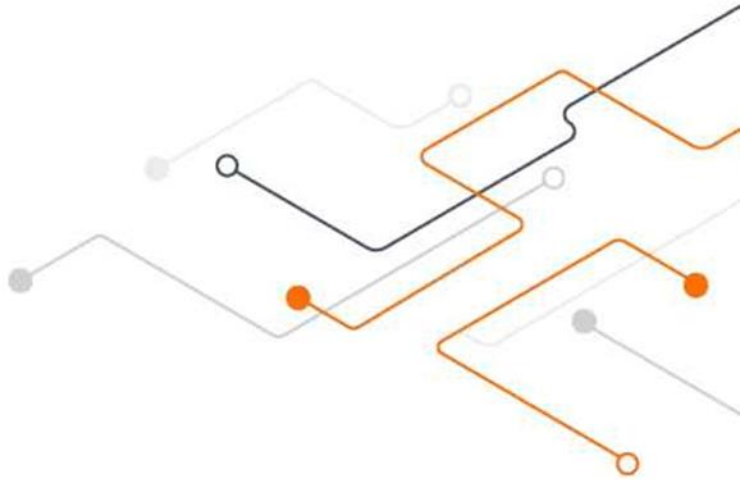
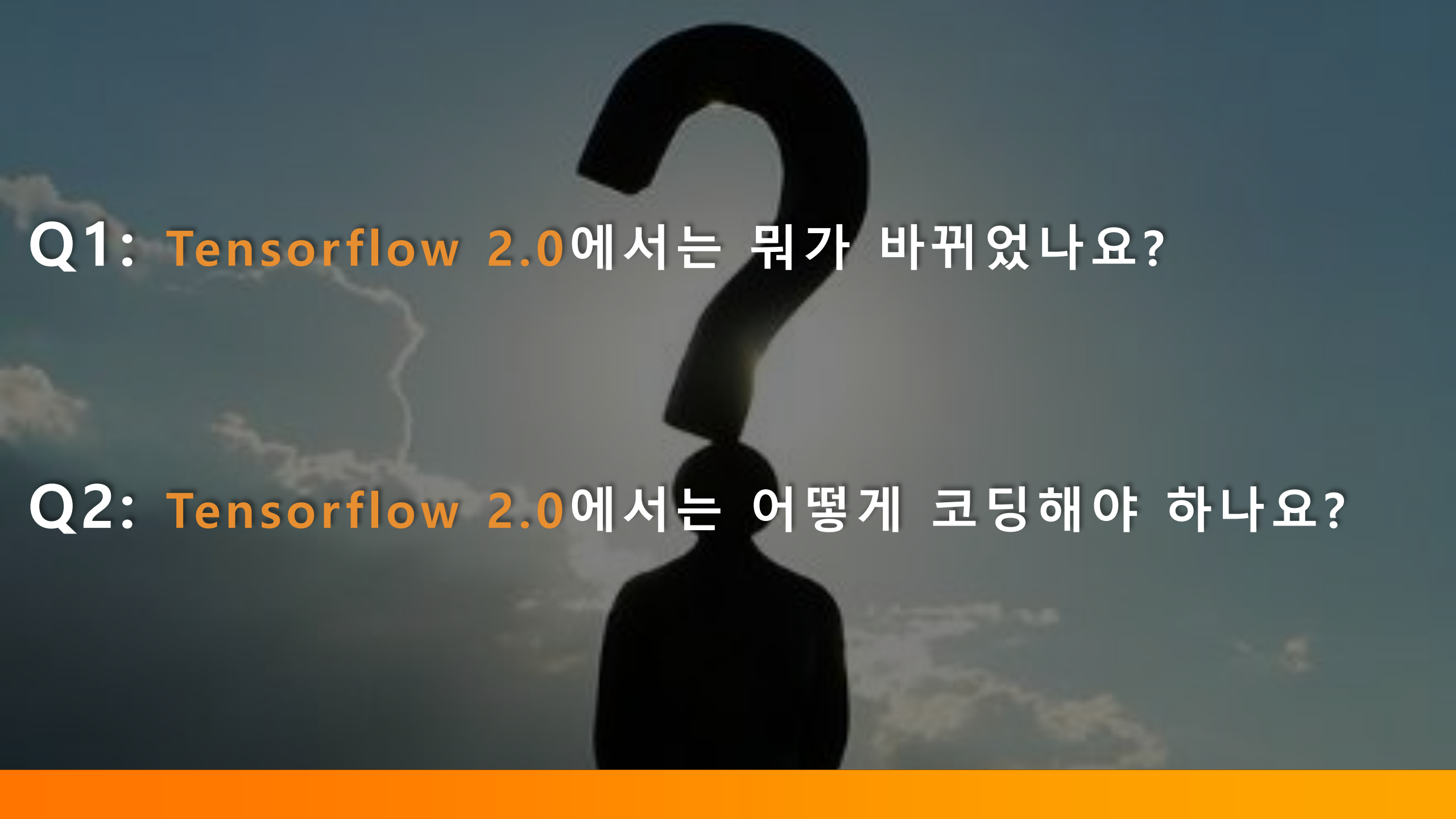
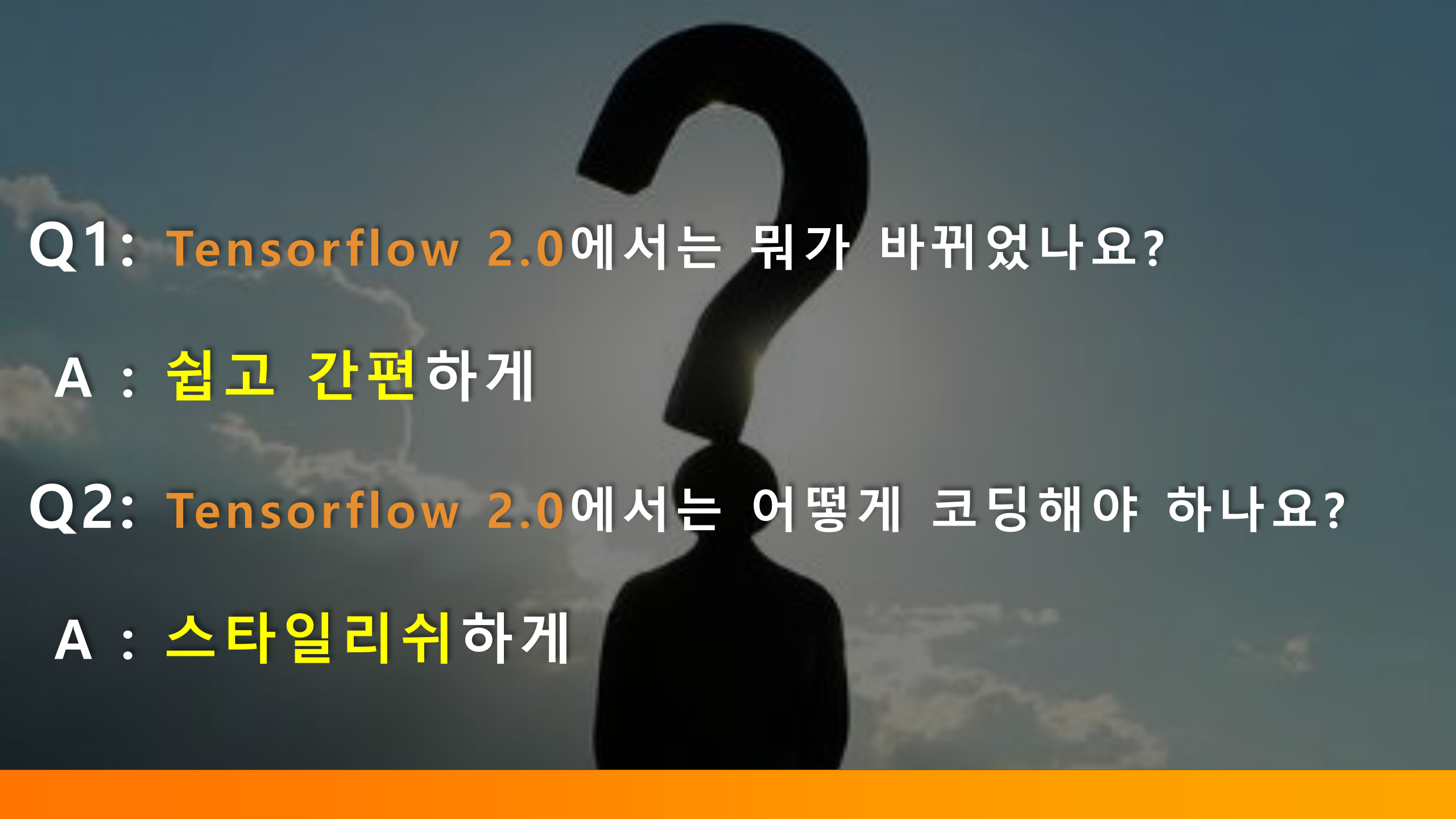
TensorFlow



Effective TF 2.0 :

Autograph and GradientTape


Q1: Tensorflow 2.0에서는 뭐가 바뀌었나요?
Q2: Tensorflow 2.0에서는 어떻게 코딩해야 하나요?

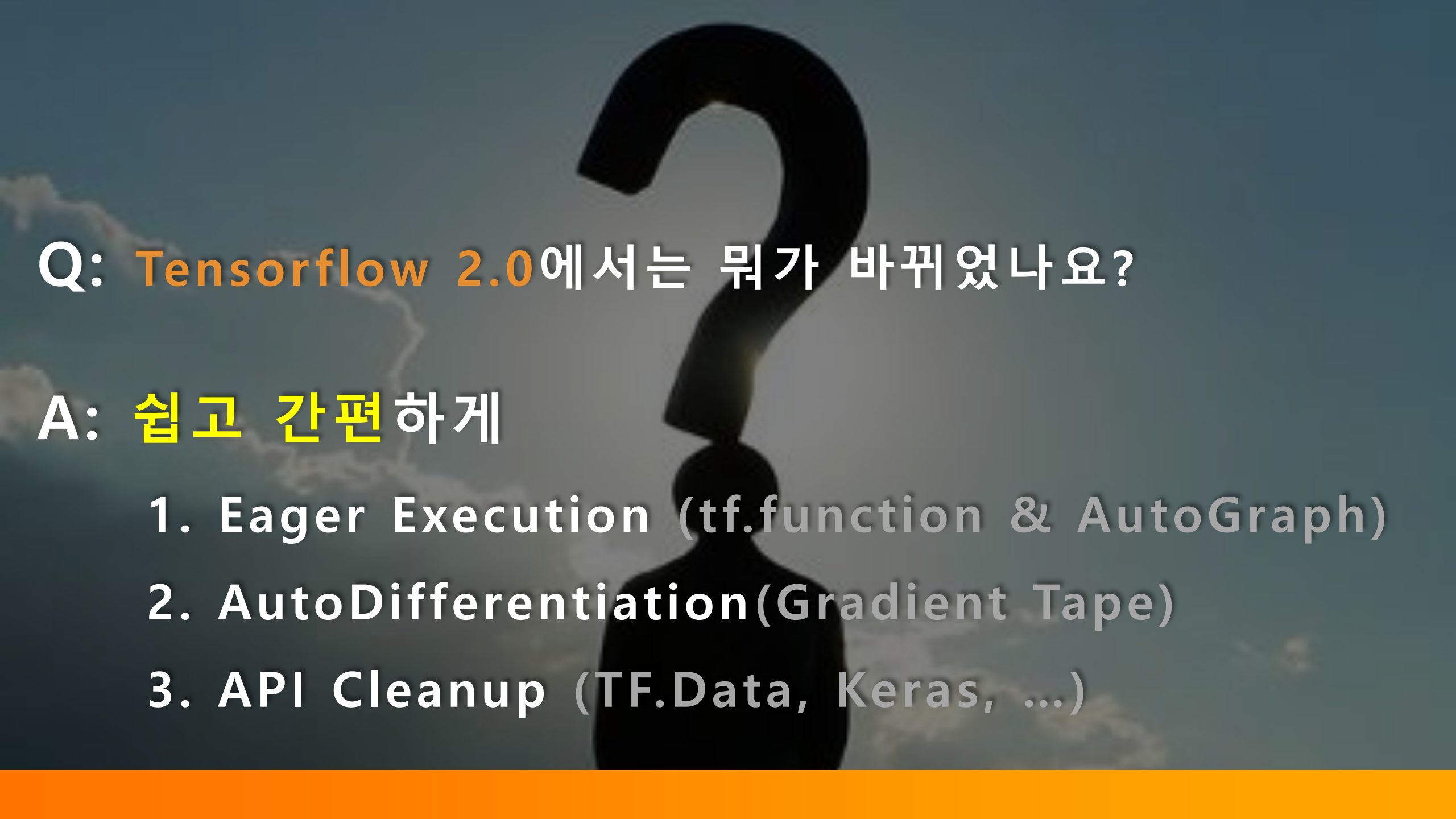


Q1: Tensorflow 2.0에서는 뭐가 바뀌었나요?

A : 쉽고 간편하게

Q2: Tensorflow 2.0에서는 어떻게 코딩해야 하나요?

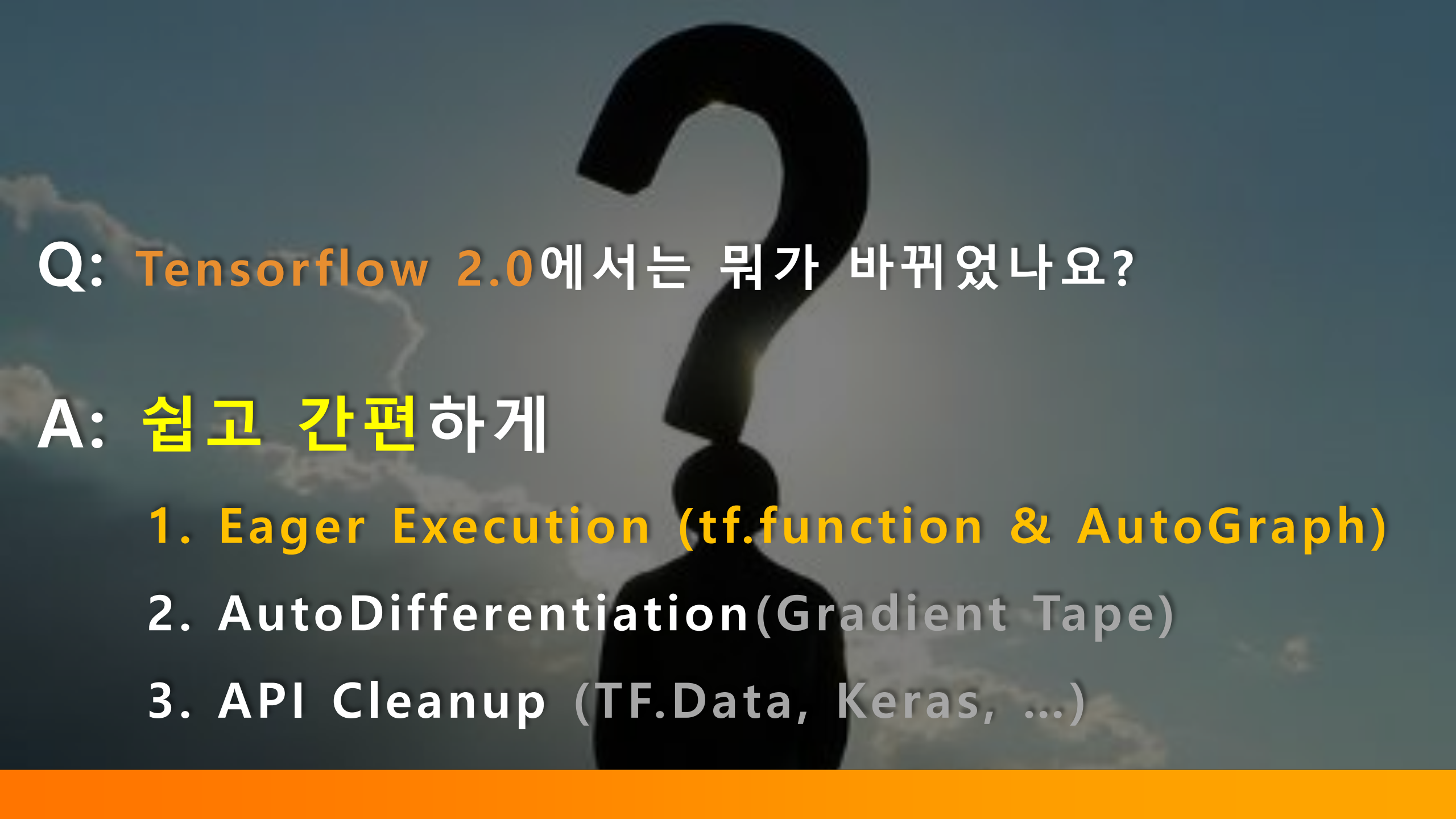
A : 스타일리쉬하게



Q: Tensorflow 2.0에서는 뭐가 바뀌었나요?

A: 쉽고 간편하게

1. Eager Execution (tf.function & AutoGraph)
2. AutoDifferentiation(Gradient Tape)
3. API Cleanup (TF.Data, Keras, ...)

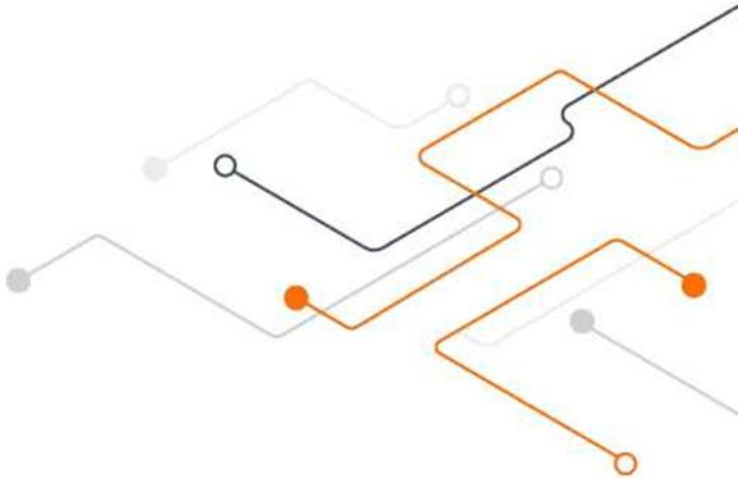


Q: Tensorflow 2.0에서는 뭐가 바뀌었나요?

A: 쉽고 간편하게

1. Eager Execution (tf.function & AutoGraph)
2. AutoDifferentiation(Gradient Tape)
3. API Cleanup (TF.Data, Keras, ...)

TensorFlow



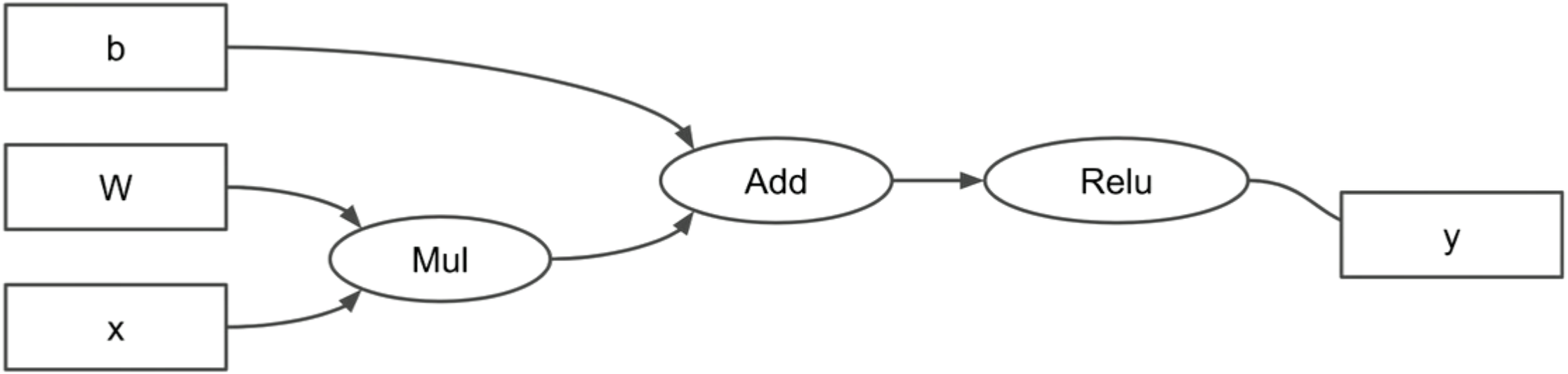
Eager Execution

Tensorflow 2.0에서는 더 이상 Session이 필요 없습니다.

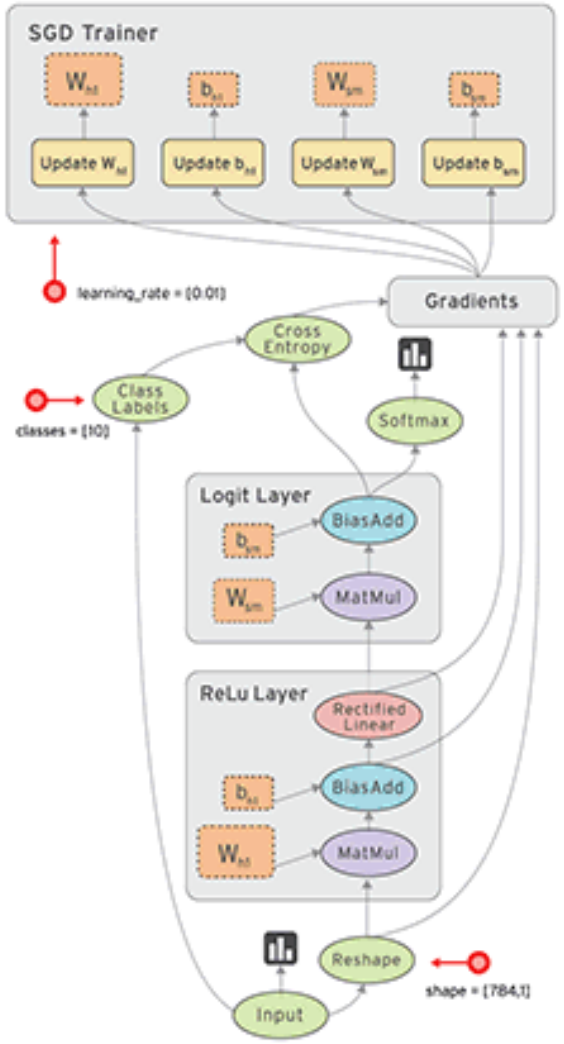
Eager Execution?

Eager execution (즉시 실행)은 대화형 명령 스타일로 프로그래밍하여 텐서플로 2.0에서는 그래프 생성 없이 연산을 즉시 실행할 수 있습니다.

What is Graph?



Tensorflow 1.0



TensorFlow 1.0: Operations are added as nodes to the computational graph and are not actually executed until we call session.run(), much like defining a function that doesn't run until it is called.

```
# 텐서플로우 1 코드

SGD_Trainer = tf.train.GradientDescentOptimizer(1e-2)

inputs = tf.placeholder(tf.float32, shape=[None, 784])
labels = tf.placeholder(tf.int16, shape=[None, 10])
hidden = ReLU_Layer(inputs)
logits = Logit_Layer(hidden)
entropy = tf.nn.softmax_cross_entropy_with_logits(
    logits=logits, labels=labels)
loss = tf.reduce_mean(entropy)
train_step = SGD_Trainer.minimize(loss,
    var_list=ReLU_Layer.weights+Logit_Layer.weights)

sess = tf.InteractiveSession()
sess.run(tf.global_variables_initializer())
for step in range(1000):
    sess.run(train_step, feed_dict={inputs:X, labels:y})
```

Graph vs Eager

Tensorflow 1 (Graph Execution)

```
x = tf.placeholder(tf.float32, shape=[1, 1])
m = tf.matmul(x, x)

with tf.Session() as sess:
  print(sess.run(m, feed_dict={x: [[2.]]}))

# Will print [[4.]]
```

Tensorflow 2 (Eager Execution)

```
x = [[2.]]
m = tf.matmul(x, x)

print(m)
```

Graph vs Eager

```
x = [[1, 2],
     [3, 4]]
m = tf.matmul(x, x)
print(m)
```

Graph vs Eager

```
x = [[1, 2],
     [3, 4]]
m = tf.matmul(x, x)
print(m)
```

Graph execution: Tensor("MatMul:0", shape = (2,2), dtype = int32)

Eager execution : tf.Tensor([[7 10]
[15 22]], shape = (2,2), dtype = int32)

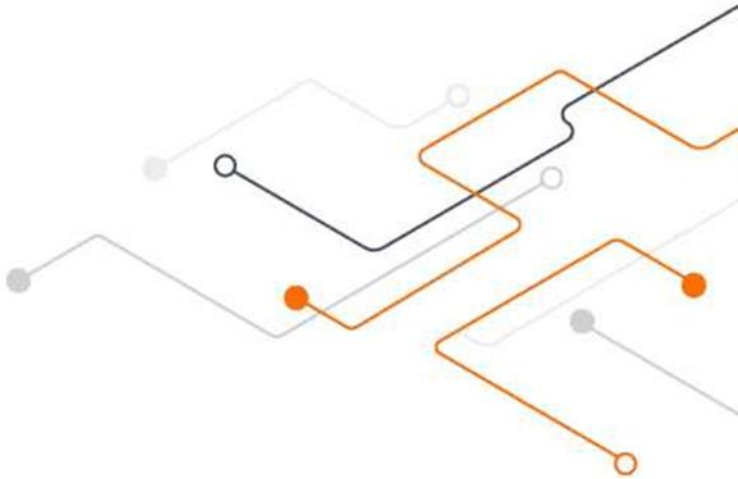
Eager Execution

```
import numpy as np

x = tf.add(1, 1)                        # tf.Tensor with a value of 2
y = tf.add(np.array(1), np.array(1)) # tf.Tensor with a value of 2
z = np.multiply(x, y)                   # numpy.int64 with a value of 4
```

```
print(y)
print(y.numpy())
```

```
tf.Tensor(2, shape=(), dtype=int64)
2
```

TensorFlow



tf.function()

Code with Eager Execution,
Run with Graphs

Autograph & tf.function

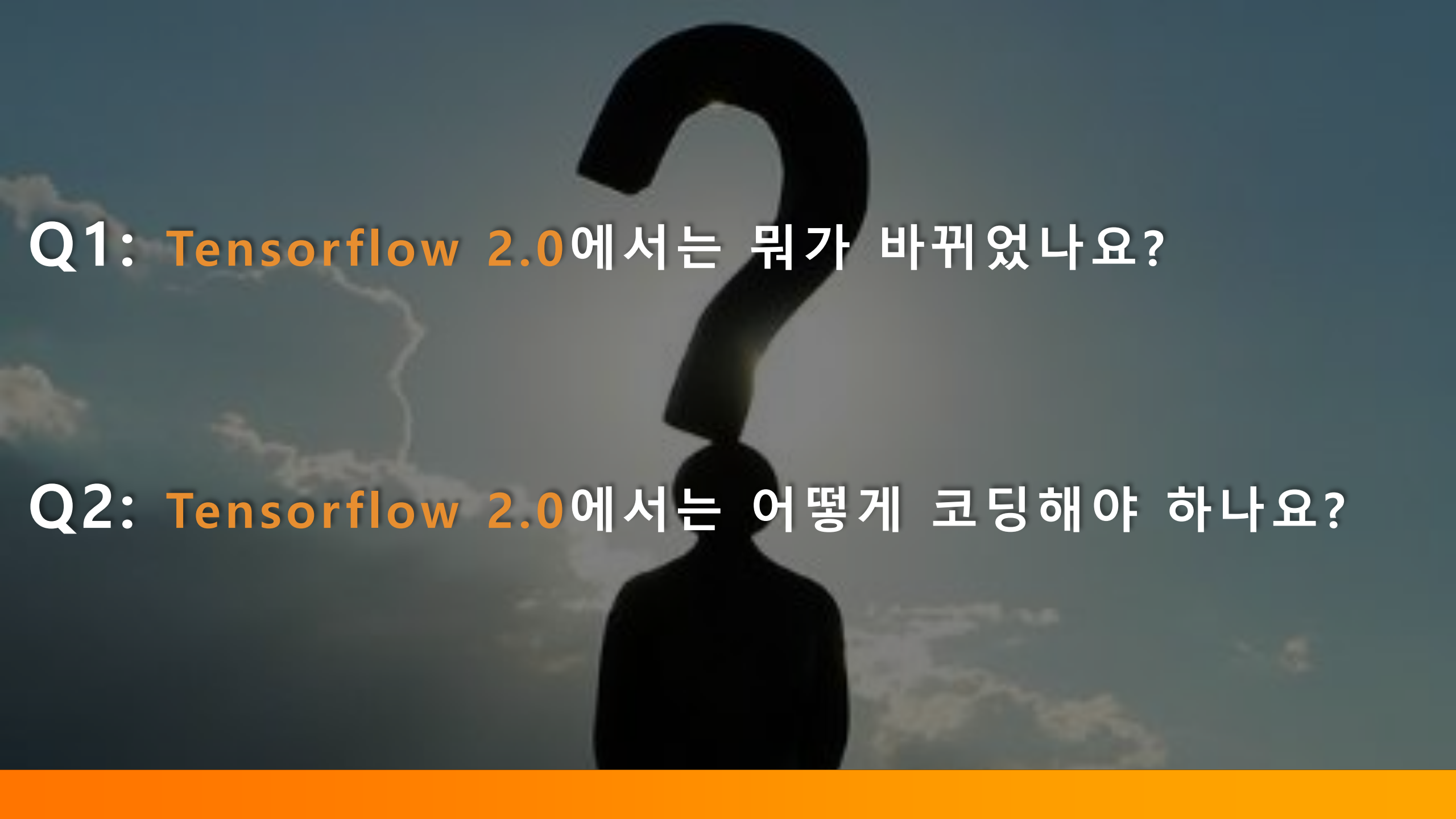
Autograph is a library that fully and deeply integrated with **@tf.function** and it will rewrite conditionals and loops(if, while, for, break, continue, etc..) which depend on **Tensors** to run dynamically in the graph

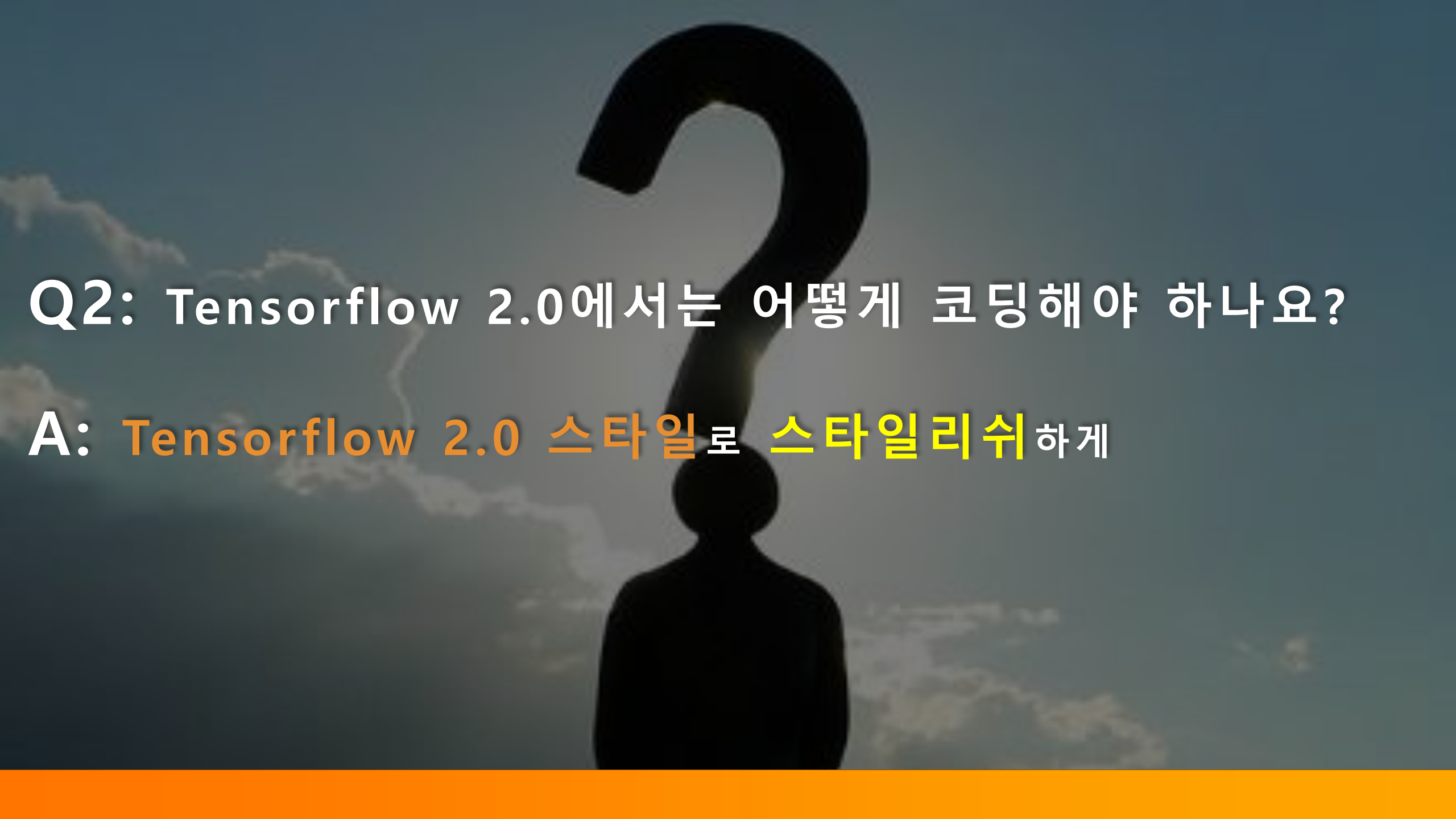
```
@tf.function
def add(a, b):
  return a + b

addf(tf.ones([2,2]), tf.ones([2,2])   #  [[2.,2.],[2.,2.]]
```

Autograph & tf.function

TensorFlow AutoGraph can convert most Python statements such as for **loops,** while **loops,** if **statements,** or **iterations.**

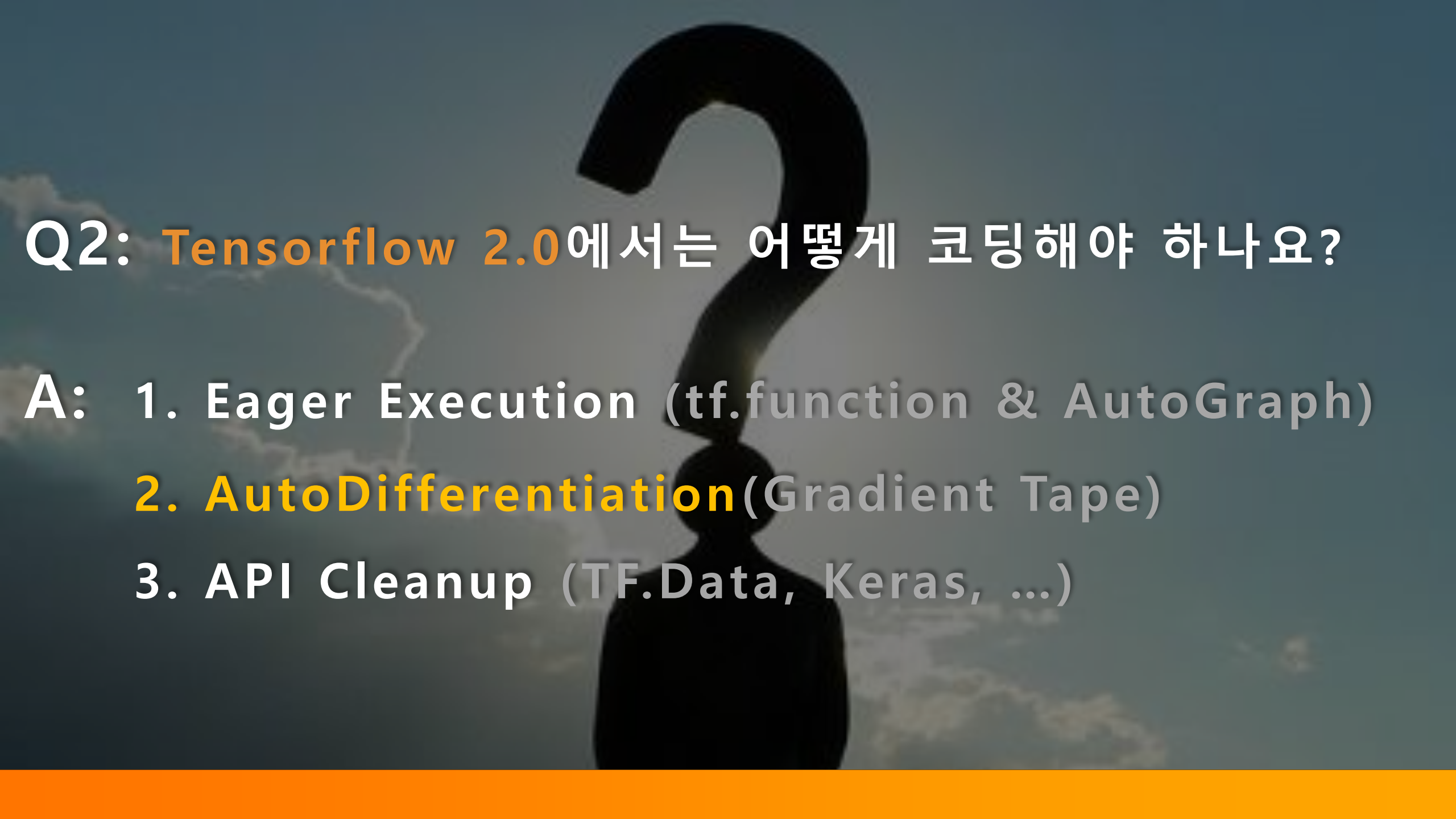
```
# Simple loop

@tf.function
def f(x):
  while tf.reduce_sum(x) > 1:
    tf.print(x)
    x = tf.tanh(x)
  return x

f(tf.random.uniform([5]))
```


Q1: Tensorflow 2.0에서는 뭐가 바뀌었나요?
Q2: Tensorflow 2.0에서는 어떻게 코딩해야 하나요?



Q2: Tensorflow 2.0에서는 어떻게 코딩해야 하나요?

A: Tensorflow 2.0 스타일로 스타일리쉬하게



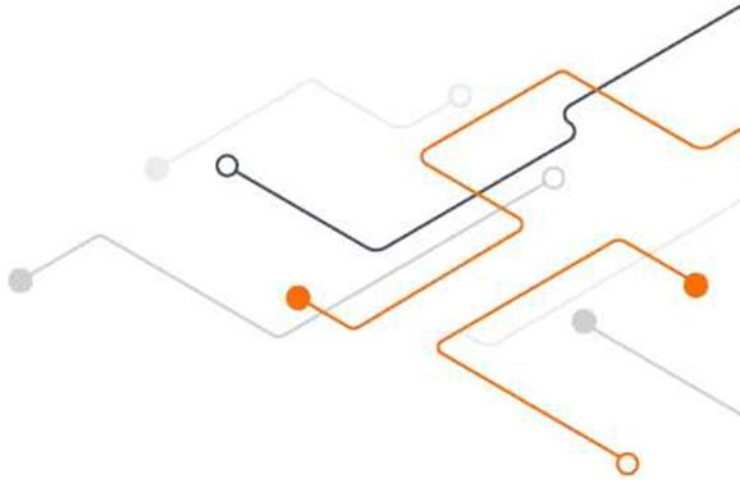
Q2: Tensorflow 2.0에서는 어떻게 코딩해야 하나요?

A: 1. Eager Execution (tf.function & AutoGraph)

2. AutoDifferentiation(Gradient Tape)

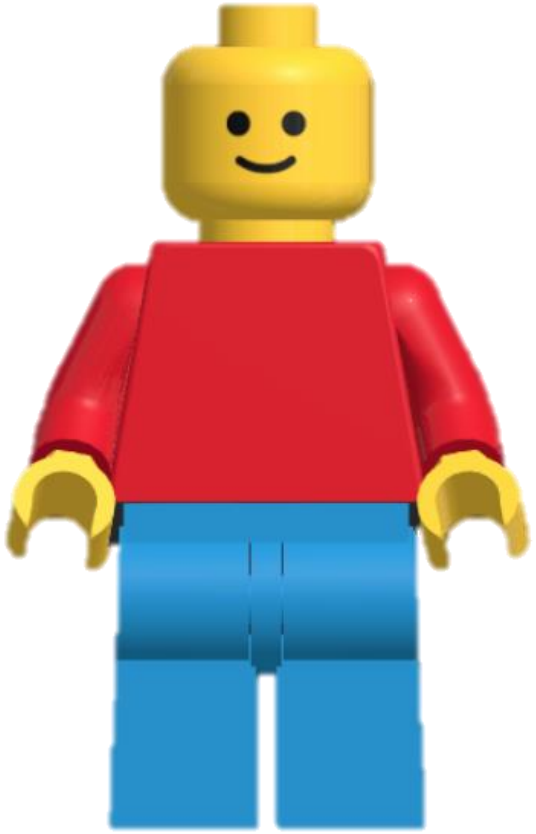
3. API Cleanup (TF.Data, Keras, ...)

TensorFlow



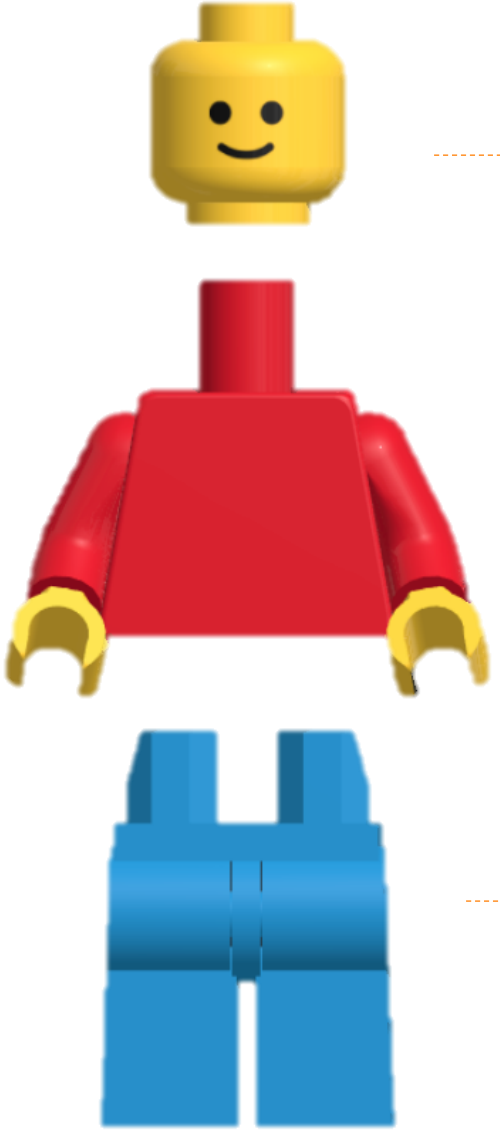
Customization

Tensorflow 2.0은 Customization을 통해 Stylish해질 수 있습니다.



Tensorflow Code

TensorFlow

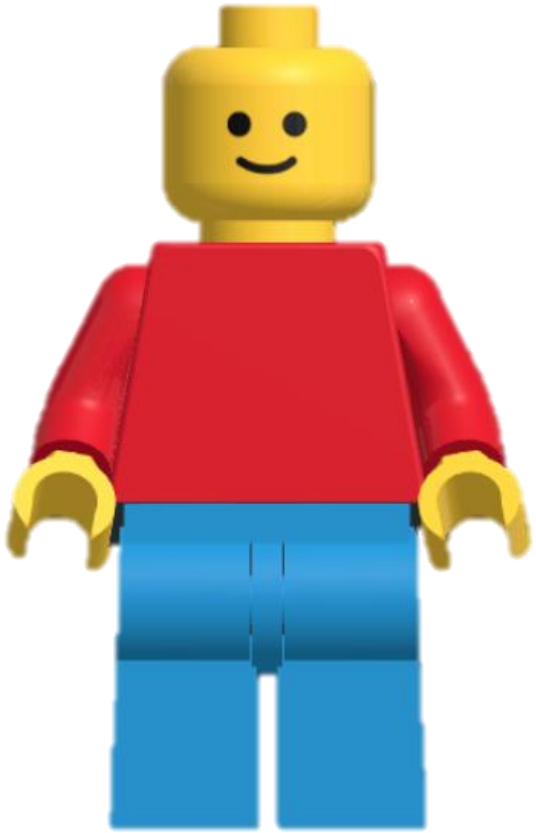


Data Input

Model

Train

TensorFlow

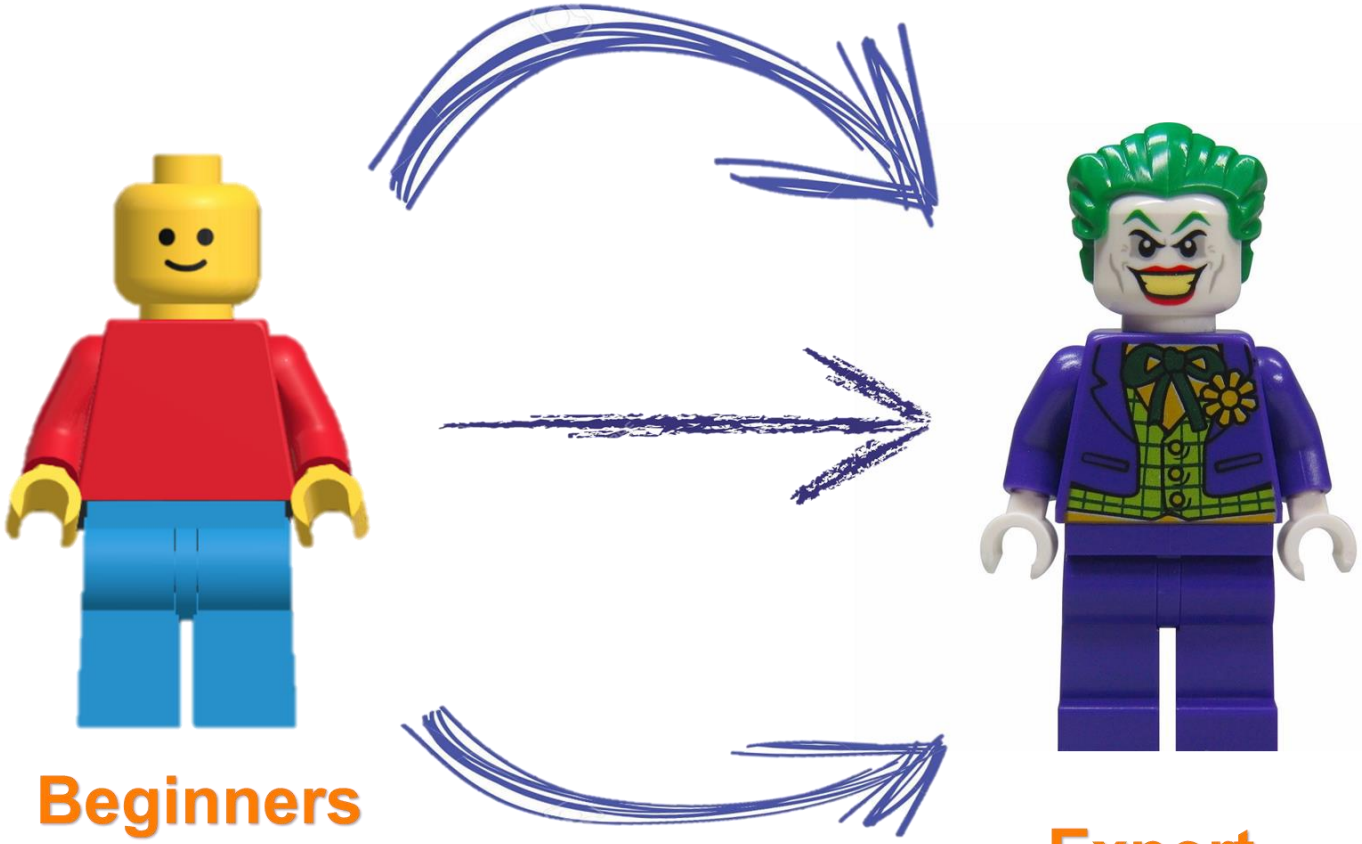


Beginners

Expert

TensorFlow

How To Customize?



```
(x_train, y_train),(x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10, activation='softmax')
])

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.fit(x_train, y_train, epochs=5)
model.evaluate(x_test, y_test)
```

Beginners

```
class MyModel(tf.keras.Model):
  def __init__(self):
    super(MyModel, self).__init__()
    self.conv1 = Conv2D(32, 3, activation='relu')
    self.flatten = Flatten()
    self.d1 = Dense(128, activation='relu')
    self.d2 = Dense(10, activation='softmax')

  def call(self, x):
    x = self.conv1(x)
    x = self.flatten(x)
    x = self.d1(x)
    return self.d2(x)
model = MyModel()

with tf.GradientTape() as tape:
  logits = model(images)
  loss_value = loss(logits, labels)
grads = tape.gradient(loss_value, model.trainable_variables)
optimizer.apply_gradients(zip(grads,
model.trainable_variables))
```

Expert

numpy

```
(x_train, y_train),(x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10, activation='softmax')
])

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.fit(x_train, y_train, epochs=5)
model.evaluate(x_test, y_test)
```

tf.keras (Sequential API)

model.compile, model.fit (keras)

Beginners

tf.data

```
class MyModel(tf.keras.Model):
  def __init__(self):
    super(MyModel, self).__init__()
    self.conv1 = Conv2D(32, 3, activation='relu')
    self.flatten = Flatten()
    self.d1 = Dense(128, activation='relu')
    self.d2 = Dense(10, activation='softmax')

  def call(self, x):
    x = self.conv1(x)
    x = self.flatten(x)
    x = self.d1(x)
    return self.d2(x)
model = MyModel()

with tf.GradientTape() as tape:
  logits = model(images)
  loss_value = loss(logits, labels)
grads = tape.gradient(loss_value, model.trainable_variables)
optimizer.apply_gradients(zip(grads,
model.trainable_variables))
```

tf.keras (functional API, Subclassing API)

tf.GradientTape, tf.function

Expert

TensorFlow

How To Customize?

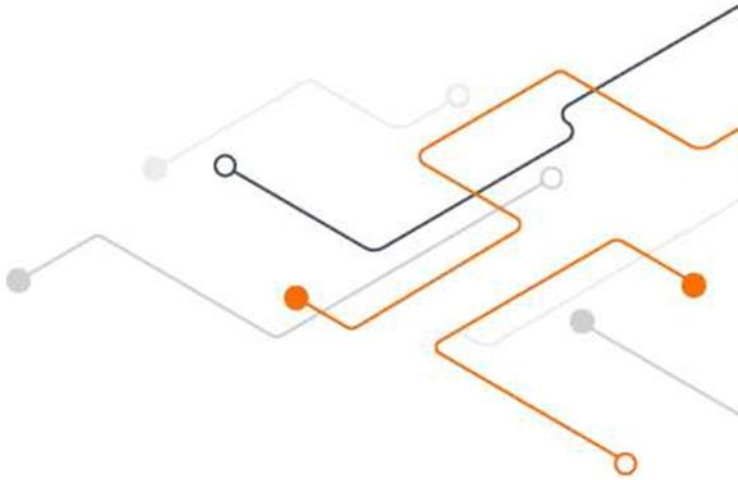
```
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.fit(x_train, y_train, epochs=5)
model.evaluate(x_test, y_test)
```

Tensorflow 2.0 Beginner

```
with tf.GradientTape() as tape:
  logits = model(images)
  loss_value = loss(logits, labels)
grads = tape.gradient(loss_value,
model.trainable_variables)
optimizer.apply_gradients(zip(grads,
model.trainable_variables
```

Tensorflow 2.0 Expert

TensorFlow



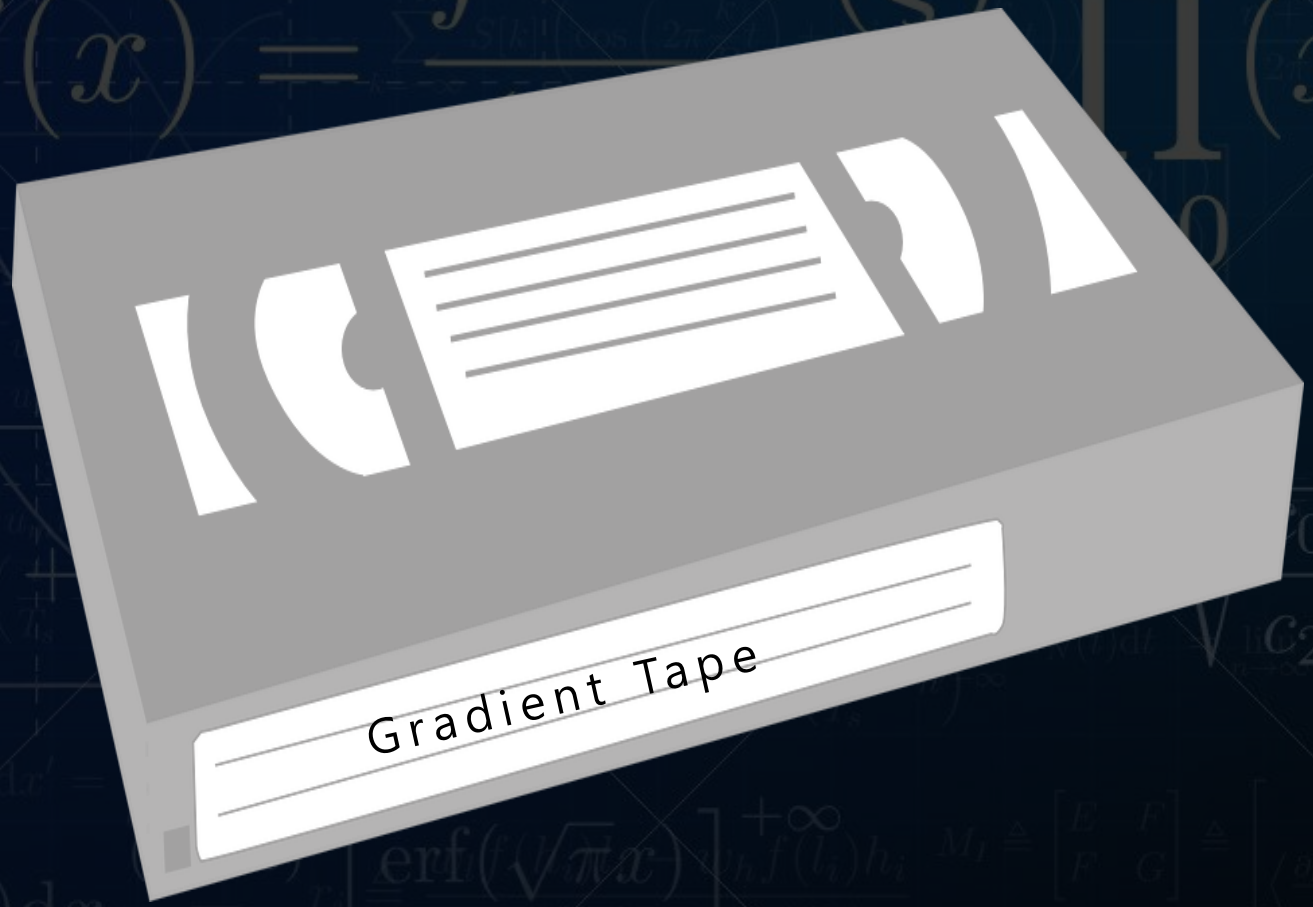
Gradient Tape

Tensorflow 2.0에 함수를 기록하는 카메라가 있습니다

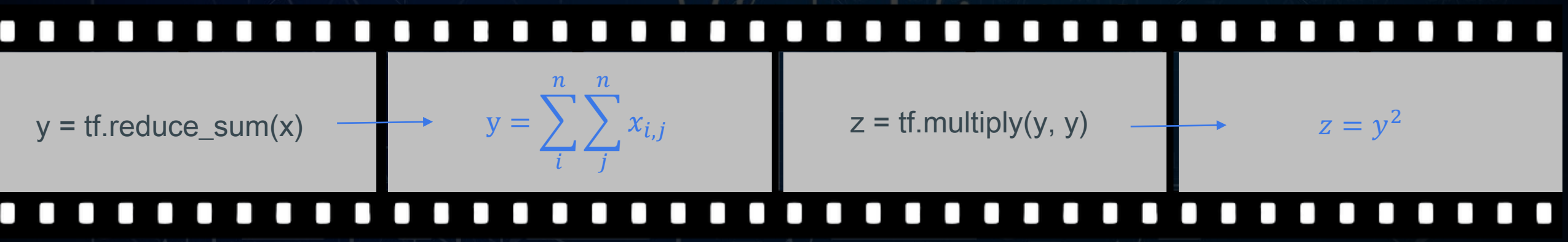
with tf.GradientTape as tape:


tape.watch
REC
00.17.02.87

GradientTape



```
with tf.GradientTape() as tape:
  tape.watch(x)
  y = tf.reduce_sum(x)
  z = tf.multiply(y, y)
```



$$\mathbf{y} = \sum_{\boldsymbol{i}}^{\boldsymbol{n}} \sum_{\boldsymbol{j}}^{\boldsymbol{n}} \boldsymbol{x_{i,j}}$$

$$\boldsymbol{z = y^2}$$

tape.gradient()

y = tf.reduce_sum(x) → $y = \sum_{i}^{n}\sum_{j}^{n} x_{i,j}$ → z = tf.multiply(y, y) → $z = y^2$

1. $y = \sum_{i}^{n}\sum_{j}^{n} x_{i,j}$ $dy/dx_{i,j} = x_{i,j}$

2. $z = y^2$ $dz/dy = 2y$

$$dz/dx_{i,j} = \frac{dz}{dy}\frac{dy}{dx_{i,j}} = 2y * x_{i,j}$$

2. $dy/dx_{i,j} = x_{i,j}$

1. $dz/dy = 2y$

Reverse

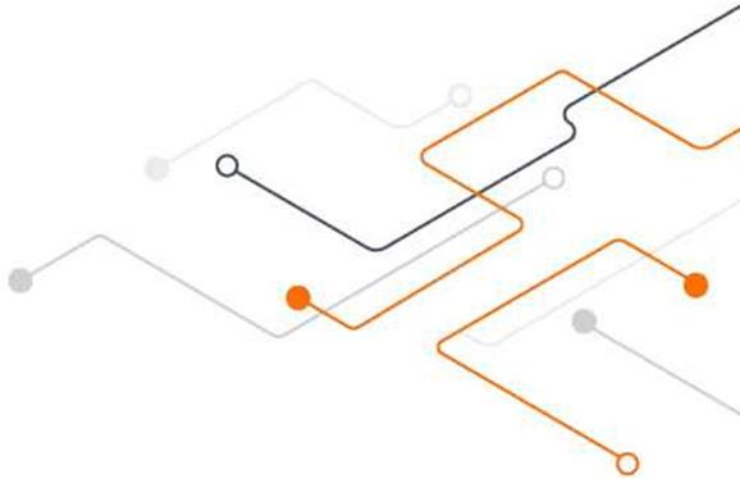
tape.gradient()

y = tf.reduce_sum(x) → $y = \sum_{i}^{n}\sum_{j}^{n} x_{i,j}$ → z = tf.multiply(y, y) → $z = y^2$

$$dz/dx_{i,j} = \frac{dz}{dy}\frac{dy}{dx_{i,j}} = 2y * x_{i,j}$$

```
with tf.GradientTape() as tape:
  tape.watch(x)
  y = tf.reduce_sum(x)
  z = tf.multiply(y, y)

dz_dx = tape.gradient(z,x)
```

TensorFlow



Custom Train

TensorFlow

Custom train

```
@tf.function
def train_step(images, labels):
  with tf.GradientTape() as tape:
    predictions = model(images)
    loss = loss_object(labels, predictions)
  gradients = tape.gradient(loss, model.trainable_variables)
  optimizer.apply_gradients(zip(gradients, model.trainable_variables))

  train_loss(loss)
  train_accuracy(labels, predictions)
```

TensorFlow

Custom train

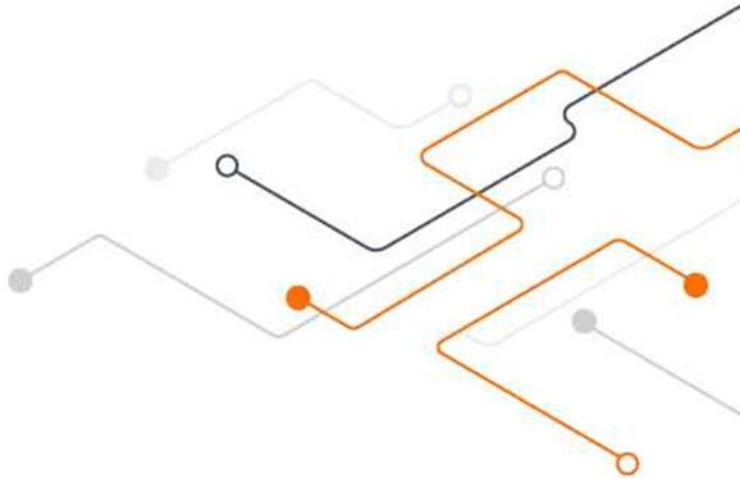
```
@tf.function
def test_step(images, labels):
  predictions = model(images)
  t_loss = loss_object(labels, predictions)


  test_loss(t_loss)
  test_accuracy(labels, predictions)
```

TensorFlow

Custom train

```
EPOCHS = 5

for epoch in range(EPOCHS):
  for images, labels in train_ds:
    train_step(images, labels)

  for test_images, test_labels in test_ds:
    test_step(test_images, test_labels)

  template = 'Epoch {}, Loss: {}, Accuracy: {}, Test Loss: {}, Test Accuracy: {}‘
  print(template.format(epoch+1,
                        train_loss.result(),
                        train_accuracy.result()*100,
                        test_loss.result(),
                        test_accuracy.result()*100))
```

TensorFlow



감사합니다.