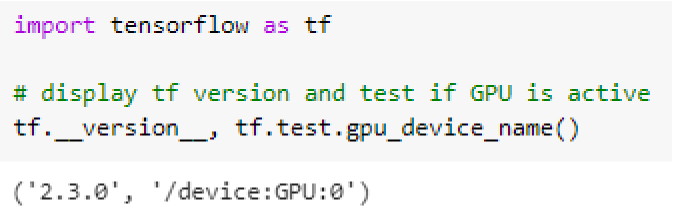
Assignment 5 (key)

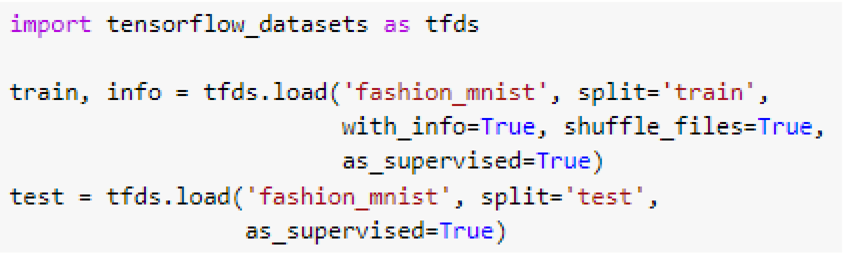
1. Enable GPU:



```
import tensorflow as tf

# display tf version and test if GPU is active
tf.__version__, tf.test.gpu_device_name()
```

```
('2.3.0', '/device:GPU:0')
```

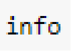
2. Load Fashion-MNIST as TFDS (**as_supervised=True**):



```
import tensorflow_datasets as tfds

train, info = tfds.load('fashion_mnist', split='train',
                        with_info=True, shuffle_files=True,
                        as_supervised=True)
test = tfds.load('fashion_mnist', split='test',
                 as_supervised=True)
```

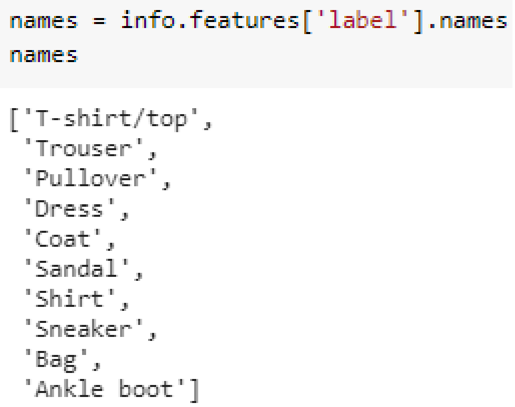
3. Inspect data.

a. display information from **info** object:



```
info
```

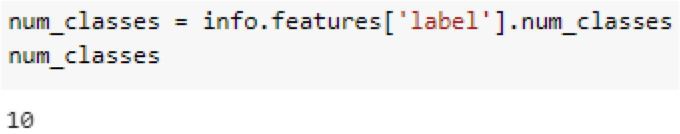
b. display class names:



```
names = info.features['label'].names
names
```

```
['T-shirt/top',
 'Trouser',
 'Pullover',
 'Dress',
 'Coat',
 'Sandal',
 'Shirt',
 'Sneaker',
 'Bag',
 'Ankle boot']
```

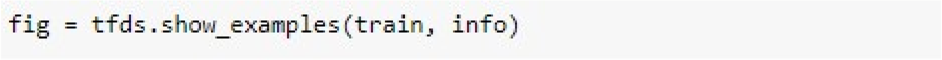
c. display number of classes:



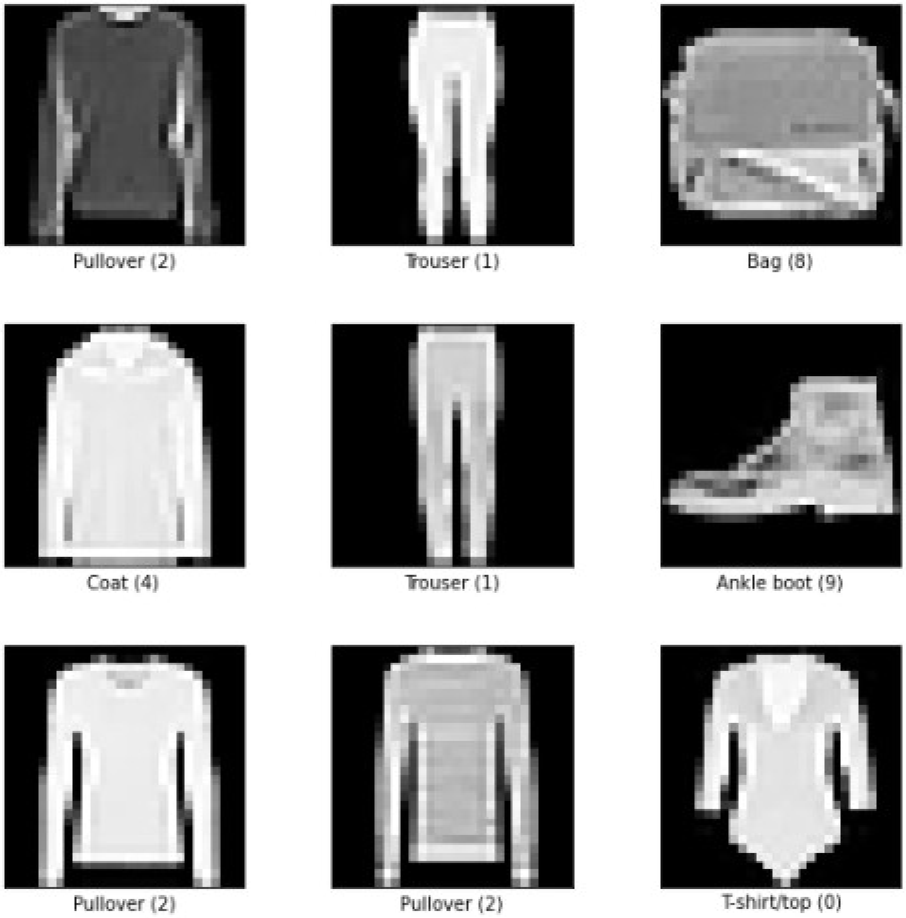
```
num_classes = info.features['label'].num_classes
num_classes
```

```
10
```

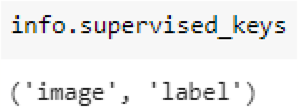
d. show some examples:



```
fig = tfds.show_examples(train, info)
```



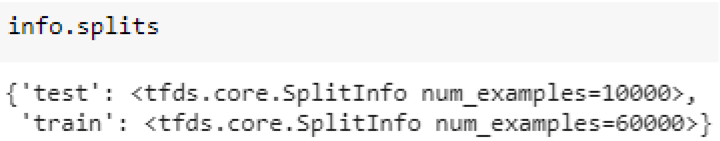
e. display supervised keys:



```
info.supervised_keys
```

```
('image', 'label')
```

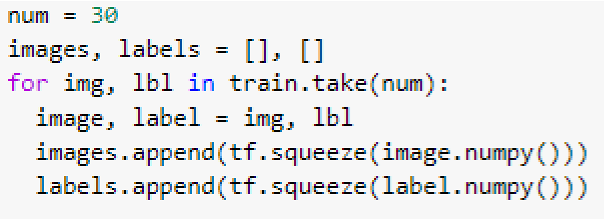
f. display splits:



```
info.splits
```

```
{'test': <tfds.core.SplitInfo num_examples=10000>,
 'train': <tfds.core.SplitInfo num_examples=60000>}
```

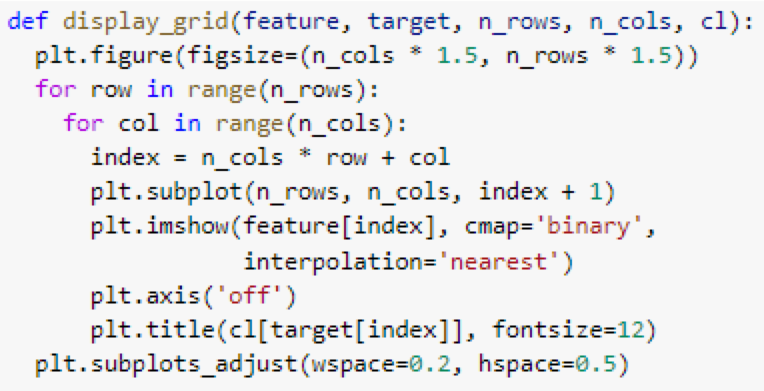
g. prepare 30 images and labels for display:



```
num = 30
images, labels = [], []
for img, lbl in train.take(num):
  image, label = img, lbl
  images.append(tf.squeeze(image.numpy()))
  labels.append(tf.squeeze(label.numpy()))
```

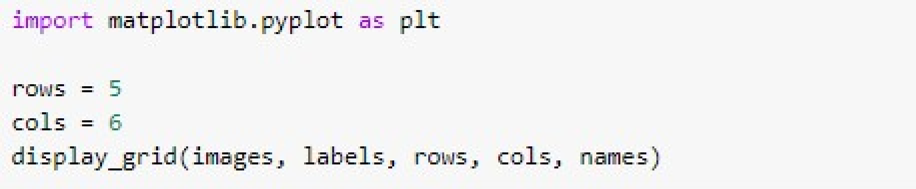
(hint: take 30 examples from **train** set, squeeze out 1 dimension, and build lists)

h. create a function to plot images and labels:

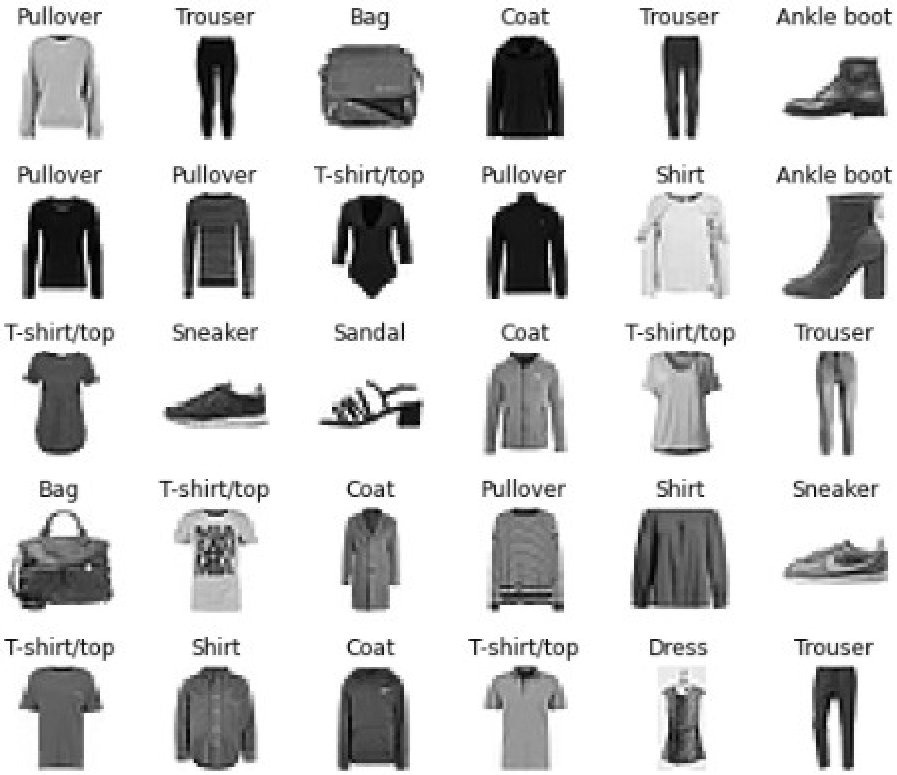


```
def display_grid(feature, target, n_rows, n_cols, cl):
  plt.figure(figsize=(n_cols * 1.5, n_rows * 1.5))
  for row in range(n_rows):
    for col in range(n_cols):
      index = n_cols * row + col
      plt.subplot(n_rows, n_cols, index + 1)
      plt.imshow(feature[index], cmap='binary',
                 interpolation='nearest')
      plt.axis('off')
      plt.title(cl[target[index]], fontsize=12)
  plt.subplots_adjust(wspace=0.2, hspace=0.5)
```

i. plot images and labels:

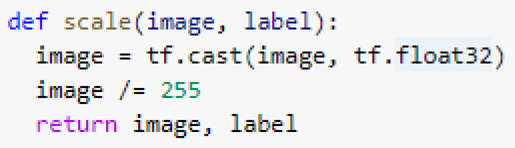


```
import matplotlib.pyplot as plt

rows = 5
cols = 6
display_grid(images, labels, rows, cols, names)
```



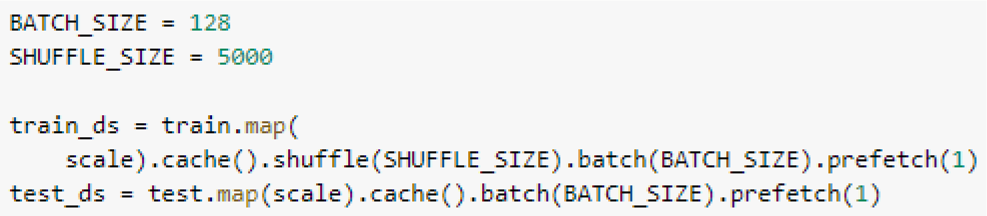
4. Build the input pipeline.

a. create a function to scale train images:



```
def scale(image, label):
  image = tf.cast(image, tf.float32)
  image /= 255
  return image, label
```

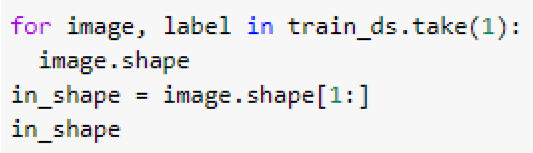
b. cache, shuffle (where appropriate), batch, and prefetch:



```
BATCH_SIZE = 128
SHUFFLE_SIZE = 5000

train_ds = train.map(
    scale).cache().shuffle(SHUFFLE_SIZE).batch(BATCH_SIZE).prefetch(1)
test_ds = test.map(scale).cache().batch(BATCH_SIZE).prefetch(1)
```

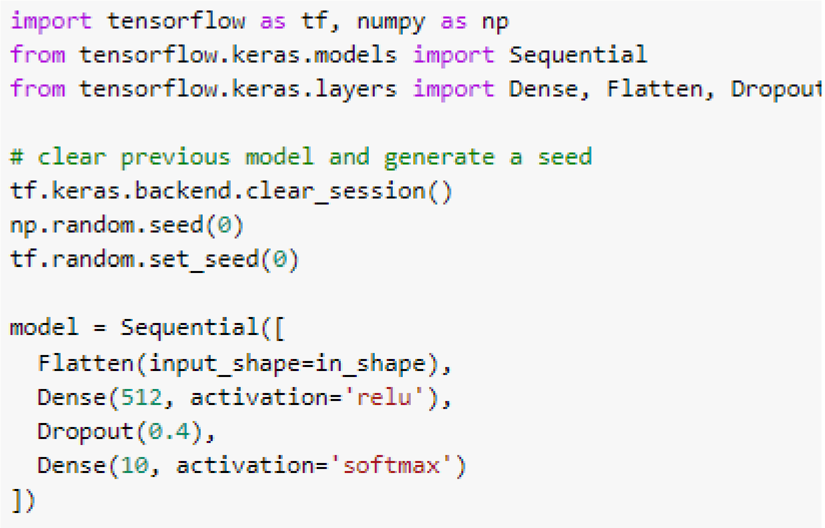
5. Build model.

a. get input shape:



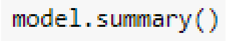
```
for image, label in train_ds.take(1):
  image.shape
in_shape = image.shape[1:]
in_shape
```

b. get libraries, clear models, generate seed, and create model:



```
import tensorflow as tf, numpy as np
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, Dropout

# clear previous model and generate a seed
tf.keras.backend.clear_session()
np.random.seed(0)
tf.random.set_seed(0)

model = Sequential([
  Flatten(input_shape=in_shape),
  Dense(512, activation='relu'),
  Dropout(0.4),
  Dense(10, activation='softmax')
])
```

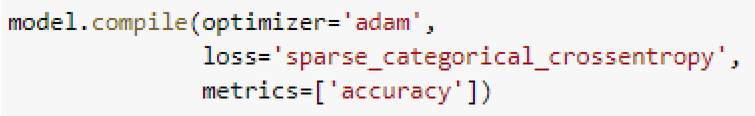
c. model summary:



```
model.summary()
```

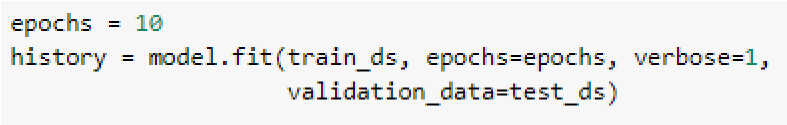
6. Train.

a. compile:



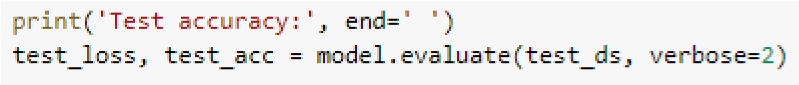
```
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

b. train for 10 epochs:



```
epochs = 10
history = model.fit(train_ds, epochs=epochs, verbose=1,
                    validation_data=test_ds)
```

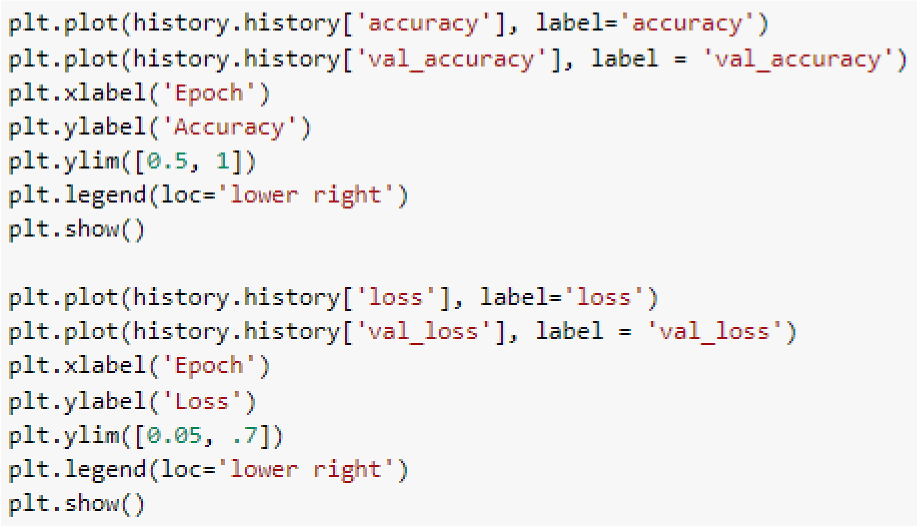
c. generalize:



```
print('Test accuracy:', end=' ')
test_loss, test_acc = model.evaluate(test_ds, verbose=2)
```

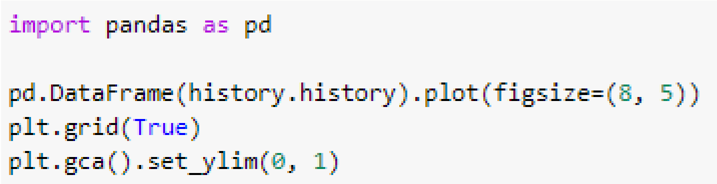
7. Visualize.

   a. plot performance:



```
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0.5, 1])
plt.legend(loc='lower right')
plt.show()

plt.plot(history.history['loss'], label='loss')
plt.plot(history.history['val_loss'], label = 'val_loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.ylim([0.05, .7])
plt.legend(loc='lower right')
plt.show()
```

   b. use pandas to plot:



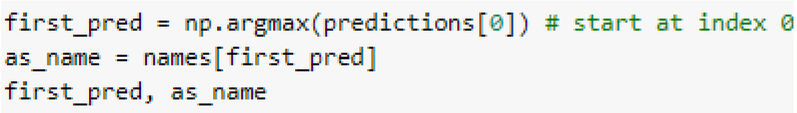
```
import pandas as pd

pd.DataFrame(history.history).plot(figsize=(8, 5))
plt.grid(True)
plt.gca().set_ylim(0, 1)
```

8. Predict.

   a. get predictions:

```
predictions = model.predict(test_ds)
```

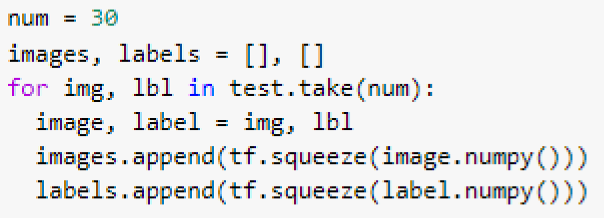
   b. get first prediction as class number and name:



```
first_pred = np.argmax(predictions[0]) # start at index 0
as_name = names[first_pred]
first_pred, as_name
```

9. Build a prediction plot.

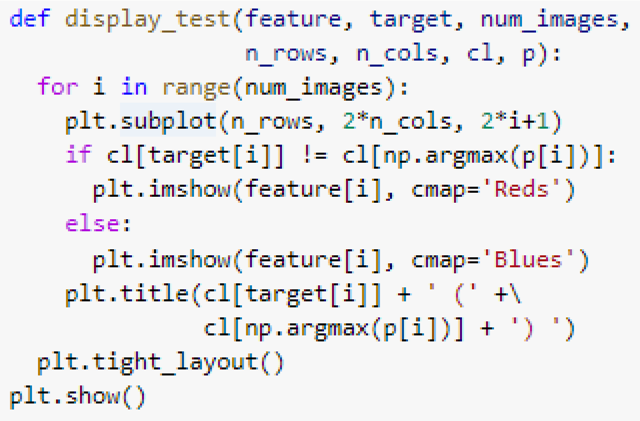
a. prepare 30 examples from the **test** set:



```
num = 30
images, labels = [], []
for img, lbl in test.take(num):
  image, label = img, lbl
  images.append(tf.squeeze(image.numpy()))
  labels.append(tf.squeeze(label.numpy()))
```

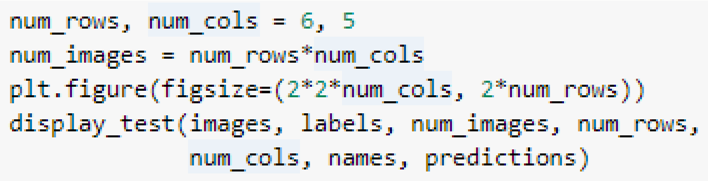
(hint: take 30 examples from test set, squeeze out 1 dimension, and build lists)

b. create a function to display:



```
def display_test(feature, target, num_images,
                 n_rows, n_cols, cl, p):
  for i in range(num_images):
    plt.subplot(n_rows, 2*n_cols, 2*i+1)
    if cl[target[i]] != cl[np.argmax(p[i])]:
      plt.imshow(feature[i], cmap='Reds')
    else:
      plt.imshow(feature[i], cmap='Blues')
    plt.title(cl[target[i]] + ' (' +\
              cl[np.argmax(p[i])] + ') ')
  plt.tight_layout()
plt.show()
```

c. display:



```
num_rows, num_cols = 6, 5
num_images = num_rows*num_cols
plt.figure(figsize=(2*2*num_cols, 2*num_rows))
display_test(images, labels, num_images, num_rows,
             num_cols, names, predictions)
```