```
import tensorflow as tf
from matplotlib import pyplot as plt
import numpy as np


objects = tf.keras.datasets.mnist
(training_images, training_labels), (test_images, test_labels) = objects.load_data()
training_images = training_images/255.0
training_images = np.expand_dims(training_images, axis=1)
train_dataset = tf.data.Dataset.from_tensor_slices((training_images, training_labels))
train_dataset = train_dataset.shuffle(buffer_size=1024).batch(100, drop_remainder=True

test_images = test_images/255.0
test_images = np.expand_dims(test_images, axis=1)
test_dataset = tf.data.Dataset.from_tensor_slices((test_images, test_labels))
test_dataset = test_dataset.shuffle(buffer_size=1024).batch(100, drop_remainder=True)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets
11493376/11490434 [==============================] - 0s 0us/step
11501568/11490434 [==============================] - 0s 0us/step
```

```
class Model(tf.keras.Model):
  def __init__(self):
    super(Model, self).__init__()
    self.dense_layer = tf.keras.models.Sequential([tf.keras.layers.Flatten(input_shape
                                tf.keras.layers.Dense(10, activation = tf.nn.softma
    self.optimizer = tf.keras.optimizers.Adam(learning_rate = 0.01)
    self.batch_size = 100
  def call(self, inputs):
    probabilities = self.dense_layer(inputs)
    return probabilities
  def loss(self, probabilities, labels):
    return tf.reduce_mean(tf.keras.metrics.sparse_categorical_crossentropy(labels, pro
  def accuracy(self, probs, labels):
    total = len(labels)
    correct = 0
    for i in range(len(probs)):
      if tf.math.argmax(probs[i]) == tf.cast(labels[i], dtype=tf.int64):
        correct += 1
    return correct/total


model = Model()


def train(model, data):
  loss = 0
  for image in data:
    inputs, labels = image
    with tf.GradientTape() as tape:
```

```
      #print(inputs.shape)
      probs = model.call(inputs)
      loss = model.loss(probs, labels)
    gradients = tape.gradient(loss, model.trainable_variables)
    model.optimizer.apply_gradients(zip(gradients, model.trainable_variables))
def test(model, data):
  accuracy = []
  for image in data:
    inputs, labels = image
    probs = model.call(inputs)
    accuracy.append(model.accuracy(probs, labels))
  return tf.reduce_mean(accuracy)

train(model, train_dataset)


print(test_dataset)
print(test(model, test_dataset))
```

```
    <BatchDataset shapes: ((100, 1, 28, 28), (100,)), types: (tf.float64, tf.uint8)>
    tf.Tensor(0.9103, shape=(), dtype=float32)
```

In [1]:
```
import turtle
import canvasvg
from turtle import Turtle, Screen
import canvasvg
from PIL import Image
screen = Screen()
t = Turtle("turtle")
t.speed(-1)
t.pensize(25)
t.shape("circle")
```

In [2]:
```
def dragging(x, y):
    t.ondrag(None)
    t.setheading(t.towards(x, y))
    t.goto(x, y)
    t.ondrag(dragging)
```

In [3]:
```
def clickright(x, y):
    saveImg()
    t.clear()
```

In [4]:
```
def main():
    turtle.listen()
    t.ondrag(dragging)
    turtle.onscreenclick(clickright, 2)
    screen.mainloop()
```

In [5]:
```
def saveImg():
    canvas = turtle.getscreen().getcanvas()
    canvas.postscript(file="turtle_img.ps")
```

In [6]:
```
main()
```