```
'''

ALGORITH TO PLOT A GRAPH FROM A CSV FILE


PLEASE NOTE:

THE CSV FILE NAME MUST BE "FinData.csv"

THE CSV FILE MUST CONTAIN THE COLUMNS "Date" (yyyy-mm-dd), "Close" (a number in either int, float, or string format)

'''


import turtle
import csv


class CSVDataHandler:
  def __init__(self, csv_file_path):
      self.csv_file_path = csv_file_path

  def csv_to_dict(self):
      #method to convert the CSV file to dictionary
      data_dictionary = {}

      with open(self.csv_file_path, mode='r') as file:  # Open CSV file for reading
          csv_reader = csv.DictReader(file)

          for row in csv_reader:  # iterate over each row in the CSV
              date = row['Date']    # access using column names
              closing_price = row['Close']

              data_dictionary[date] = closing_price  # add them to the dictionary

      return data_dictionary


class TurtleGraph:
    # a class to represent and plot a graph using turtle graphics
    def __init__(self, screen_size_x=1000, screen_size_y=1000, title="Price - Time Graph"):
        # initialise the graph with a screen size and a title
        self.screen_size_x = screen_size_x
        self.screen_size_y = screen_size_y
        self.screen = self.initialise_screen()
        self.turtle = self.initialise_turtle()
        self.screen.title(title)

    def initialise_screen(self):
        # initialise the screen
        screen = turtle.Screen()
        screen.screensize(self.screen_size_x, self.screen_size_y)
        return screen

    def initialise_turtle(self):
```

```python
    t = turtle.Turtle()
    t.speed(0) # fastest drawing speed
    t.width(2) # line width
    return t

  def find_axis_lenghts(self):
   # the subrouine name tells itself what it does
   return [self.screen_size_x * 5 / 11, self.screen_size_y * 5 / 11]

  def draw_axis(self):
    # Calculate axis lengths and positions
    axis_length_x, axis_length_y = self.find_axis_lenghts()
    arrow_size = 0.005  # Relative size of the arrow head

    # Draw X and Y axes with arrows
    self.draw_line_with_arrows(0, 0, axis_length_x, 0, arrow_size)
    self.draw_line_with_arrows(0, 0, 0, axis_length_y, arrow_size, vertical=True)

  def draw_line_with_arrows(self, start_x, start_y, end_x, end_y, arrow_size, vertical=False):
    self.turtle.penup()
    self.turtle.goto(start_x, start_y)
    self.turtle.pendown()
    self.turtle.goto(end_x, end_y)

    # Draw arrow head
    if vertical:
     self.draw_arrow_head(end_x, end_y, arrow_size, vertical=True)
    else:
     self.draw_arrow_head(end_x, end_y, arrow_size)

    self.turtle.penup()
    self.turtle.setpos(0,0)

  def draw_arrow_head(self, x, y, size, vertical=False):
    # Draw an arrow head at specified position
    adjust_x = self.screen_size_x * size
    adjust_y = self.screen_size_y * size
    if vertical:
      self.turtle.goto(x - adjust_x, y - adjust_y)
      self.turtle.goto(x, y)
      self.turtle.goto(x + adjust_x, y - adjust_y)
    else:
      self.turtle.goto(x - adjust_x, y + adjust_y)
      self.turtle.setposition(x, y)
      self.turtle.goto(x - adjust_x, y - adjust_y)

  def find_number_of_x_axis_points(self, dict):
   return len(dict)

  def plot_graph(self, data_points):
   '''
   Please note: the instructions to calculate the x-coordinate of the points are
   commented out in the code, as dates instead of numerical values will be used on
   the x-axis.  However, the code can still be used if there is a need to plot a
```

```python
      graph with numerical values for the x-axis.
      '''

      # comment out the x axis (keys)

      # find the lowest key in the dictionary
      # lowest_key = self.find_min_or_max_key_in_dictionary(data_points) #
      # find the highest key in the dictionary
      # highest_key = self.find_min_or_max_key_in_dictionary(data_points, False) #
      # find the lowest value in the dictionary
      lowest_value = self.find_min_or_max_value_in_dictionary(data_points)
      # find the highest value in the dictionary
      highest_value = self.find_min_or_max_value_in_dictionary(data_points, False)

      dist_between_the_points_on_x = (self.find_axis_lenghts()[0]) / self.find_number_of_x_axis_points(data_points)

      k = 0 # counter for the x-axis

      for x_value, y_value in data_points.items():
          x_real_to_label = str(x_value)
          y_real_to_label = float(y_value)

          x_relative_to_plot = k * dist_between_the_points_on_x
          y_relative_to_plot = (y_real_to_label - lowest_value) / (highest_value)*(self.find_axis_lenghts()[1])

          # Plot the actual point
          self.turtle.pendown()
          self.turtle.goto(x_relative_to_plot, y_relative_to_plot)

          # Label the point on the x-axis
          self.label_point_x(x_relative_to_plot, x_real_to_label)

          # Label the point on the y-axis
          self.label_point_y(y_relative_to_plot, y_real_to_label)

          # Prepare for the next point
          self.turtle.penup()
          self.turtle.goto(x_relative_to_plot, y_relative_to_plot)

          k += 1  # increment the counter for the x-axis


    def label_point_x(self, x, x_value):
      # Label the point on the x axis
      self.turtle.penup()
      self.turtle.goto(x, 0)
      self.turtle.dot(5)  # Mark the point on the x axis
      self.turtle.goto(x, -20)
      self.turtle.pendown()
      self.turtle.write(x_value, align="center")  # Center align the x axis label

    def label_point_y(self, y, y_value):
      # Label the point on the y-axis
      self.turtle.penup()
      self.turtle.goto(0, y)
      self.turtle.dot(5)  # Mark the point on the y axis
      self.turtle.goto(-(self.screen_size_x /20), (y-y*0.05))
      self.turtle.pendown()
      self.turtle.write(y_value, align="center")  # Center align the y axis label

    def find_min_or_max_key_in_dictionary(self, data_points, minimum=True):
      return float(min(data_points.keys()) if minimum else max(data_points.keys()))

    def find_min_or_max_value_in_dictionary(self, data_points, minimum=True):
      return float(min(data_points.values()) if minimum else max(data_points.values()))

    def finish(self):
      #clean up by hiding the turtle and displaying the plot until closed
      turtle.done()
      self.turtle.hideturtle()

if __name__ == "__main__":
    graph = TurtleGraph()
    graph.draw_axis()
    csv_file = 'FinData.csv'
    data_handler = CSVDataHandler(csv_file)
    data_in_dict = data_handler.csv_to_dict()
    graph.plot_graph(data_in_dict)
    graph.finish()










# import turtle

# class TurtleGraph:
#     def __init__(self, screen_size_x=2000, screen_size_y=2000, title="Data Points Plotter"):
#         self.screen_size_x = screen_size_x
#         self.screen_size_y = screen_size_y
#         self.screen = self.initialise_screen()
#         self.turtle = self.initialise_turtle()
#         self.screen.title(title)

#     def initialise_screen(self):
#         screen = turtle.Screen()
#         screen.screensize(self.screen_size_x, self.screen_size_y)
#         return screen

#     def initialise_turtle(self):
#         t = turtle.Turtle()
#         t.speed(0)
#         t.width(2)
#         return t

#     def draw_axis(self):
#         # Calculate axis lengths and positions
#         axis_length_x = self.screen_size_x * 5 / 11
#         axis_length_y = self.screen_size_y * 5 / 11
#         arrow_size = 0.01  # Relative size of the arrow head

#         # Draw X axis
#         self.draw_line_with_arrows(-axis_length_x, -axis_length_y, axis_length_x, -axis_length_y, arrow_size)

#         # Draw Y axis
#         self.draw_line_with_arrows(-axis_length_x, -axis_length_y, -axis_length_x, axis_length_y, arrow_size, vertical=True)

#     def draw_line_with_arrows(self, start_x, start_y, end_x, end_y, arrow_size, vertical=False):
#         self.turtle.penup()
#         self.turtle.goto(start_x, start_y)
#         self.turtle.pendown()
#         self.turtle.goto(end_x, end_y)

#         # Draw arrow head
#         if vertical:
#             self.turtle.goto(end_x - self.screen_size_x * arrow_size, end_y - self.screen_size_y * arrow_size)
#             self.turtle.goto(end_x, end_y)
#             self.turtle.goto(end_x + self.screen_size_x * arrow_size, end_y - self.screen_size_y * arrow_size)
#         else:
#             self.turtle.goto(end_x - self.screen_size_x * arrow_size, end_y + self.screen_size_y * arrow_size)
#             self.turtle.setposition(end_x, end_y)
#             self.turtle.goto(end_x - self.screen_size_x * arrow_size, end_y - self.screen_size_y * arrow_size)

#         self.turtle.penup()
#         self.turtle.setpos(0,0)


#     def plot_graph(self, data_points):
#       for x_value, y_value in data_points.items():
#           x = float(x_value)
#           y = float(y_value)

#           # Plot the actual point
#           self.turtle.pendown()
#           self.turtle.goto(x, y)

#           # Label the point on the x-axis
#           self.label_point_x(x, -10, x_value)

#           # Label the point on the y-axis
#           self.label_point_y(-20, y, y_value)

#           # Prepare for the next point
#           self.turtle.penup()
#           self.turtle.goto(x, y)
#     def label_point_x(self, x, y, x_value):
#       # Label the point on the x axis
#       self.turtle.penup()
#       self.turtle.goto(x, 0)
#       self.turtle.dot(5)  # Mark the point on the x axis
#       self.turtle.goto(x, -20)
#       self.turtle.pendown()
#       self.turtle.write(x_value, align="center")  # Center align the x axis label

#     def label_point_y(self, x, y, y_value):
#       # Label the point on the y-axis
#       self.turtle.penup()
#       self.turtle.goto(0, y)
#       self.turtle.dot(5)  # Mark the point on the y axis
#       self.turtle.goto(-20, y)
#       self.turtle.pendown()
#       self.turtle.write(y_value, align="center")  # Center align the y axis label

#     def finish(self):
#       #clean up by hiding the turtle and displaying the plot until closed
#       turtle.done()
#       self.turtle.hideturtle()

# if __name__ == "__main__":
#     graph = TurtleGraph()
#     graph.draw_axis()
#     data_points = {"100": "100", "140": "180", "150": "250" , "200": "350", "400": "450"}
#     graph.plot_graph(data_points)
#     graph.finish()



# def plot_graph(data_points):
#     """
#     Plot and label a graph based on a dictionary of data points.

#     Args:
#         data_points (dict): A dictionary where keys are x-axis values and values are y-axis values.
#     """

#     # Ensure turtle graphics window is open and turtle is initialised
#     turtle_screen = turtle.Screen()
#     turtle_screen.title("Data Points Plotter")
#     plotter = turtle.Turtle()
#     plotter.speed(0)  # drawing speed

#     for x_value, y_value in data_points.items():
#         # Convert string to float for plotting
#         x = float(x_value)
#         y = float(y_value)

#         # Plot the actual point on the graph
#         plotter.pendown()
#         plotter.goto(x, y)

```

```python

#         # Label the point on the x-axis
#         plotter.penup()
#         plotter.goto(x, 0)
#         plotter.dot(5)  # Mark the point on the x axis
#         plotter.goto(x, -20)
#         plotter.pendown()
#         plotter.write(x_value, align="center")  # Center align the x-axis label

#         # Label the point on the y-axis
#         plotter.penup()
#         plotter.goto(0, y)
#         plotter.dot(5)  # Mark the point on the y axis
#         plotter.goto(-20, y)
#         plotter.pendown()
#         plotter.write(y_value, align="center")  # Right align the y-axis label

#         plotter.penup()  # Prepare for the next point
#         plotter.goto(x, y)

#     plotter.hideturtle()
#     turtle.done()

# Example usage
# data_points = {"100": "100", "140": "180", "150": "250" , "200": "350", "400": "450"}
# plot_graph(data_points)

```