## main.py

```
#################################################
# Welcome to our company valuation Python code !
#################################################


#  The following code is divided in two part.
#  Part 1 is about coding the GUI.
#  Part 2 is about the DCF model.


##################################
# Part 1: Coding the GUI
##################################

# As a first step, one should install the package tkinter, it enables us to simply create GUIs.
# Run the following line in the terminal:  "pip install tk"

#Imoprting the tkinter package. Rename it as tk.

import tkinter as tk

#Creating a root widget to initialize tkinter

root = tk.Tk()

# Giving a title to our GUI and dimensions

root.title("DCF Model GUI")
root.geometry("1400x700")


#################
# Tkinter Labels
#################

# This section is about Labels. Labels are boxes of text that we can display on our GUIS.
# We can choose the font, the font style, and font size in the font section of the Label.
# When a label is created, we can use the grid function to position the object on the GUI.
# We inserted blank labels to create some spacing between objects on the screen. This is required
# because otherwise the objects would appear close to each others if no other objects are in the way.

# Here we create three labels by using the Label function. We then position them on the screen using the
# grid system. We can specify the row and the column where we want to place the object.

# Title Label
# We give our label a name (title_label). Then call the function Label from the package to create a label.
# The "root" word means we want to create the element in the initial root.
# We can specify the font by passing different parameters.
title_label = tk.Label(root, text="Discounted Free Cash Flow Valuation Model", font=("calibre", 20, "bold"))
title_label.grid(row=0, column=0)


#Welcome Label
welcome_label = tk.Label(root, text="Welcome to our python valuation model !", font=("calibre", 13))
welcome_label.grid(row=1, column=0)


#Blank number 0
blank = tk.Label(root, text="                                ", font=("calibre", 20, "bold"))
blank.grid(row=2, column=0)


#Ticker symbol Label
ticker_symbol_box = tk.Label(root, text="1) Enter the ticker symbol (US stock):", font=("times new roman", 14))
ticker_symbol_box.grid(row=6, column=0)


#Blank number 1
blank_1 = tk.Label(root, text="                        ", font=("calibre", 20, "bold"))
blank_1.grid(row=6, column=1)


#Stock price output Label
output_stock_price = tk.Label(root, text="Actual Stock Price: ", font=("arial", 14))
output_stock_price.grid(row=6, column=2)


#Past growth Label
output_growth_rate_fcf = tk.Label(root, text="4-yr growth rate of FCF: ", font=("arial", 14))
output_growth_rate_fcf.grid(row=7, column=2)
```

```
#Future growth rate Label
future_growth_rate_fcf = tk.Label(root, text="2) Future growth rate assumption of FCF in %: ", font=("times new roman", 14))
future_growth_rate_fcf.grid(row=16, column=0)


#Discount rate Label
future_discount_rate = tk.Label(root, text = "Discount rate (desired returns) in %: ", font=("times new roman", 14))
future_discount_rate.grid(row=18, column=0)


#Blank number 2
blank_2 = tk.Label(root, text= "            ", font=("calibre", 20, "bold"))
blank_2.grid(row=13, column=2)


#Calculated value Label
output_intrinsic_value = tk.Label(root, text="Intrinsic Value: ", font=("arial", 14))
output_intrinsic_value.grid(row= 14, column=2)


#Output potential Label
output_upside_potential = tk.Label(root, text="Upside potential: ", font=("arial", 14))
output_upside_potential.grid(row=16, column=2)

# All the texts elements are now displayed on the screen.



#################
# Tkinter Inputs
#################

# Here we create and position the user input elements.
# User input are created by using the entry function.
# We can vary the width of the entry box by changing the number.
# Same as for labels, we can position the element by using the grid
# function.

#Ticker symbol from user's input
input_ticker_symbol = tk.Entry(root, width=12)
input_ticker_symbol.grid(row=6, column=1)


#Future growth rate from user's input
input_future_growth_rate = tk.Entry(root, width=12)
input_future_growth_rate.grid(row=16, column=1)


#Discount rate from user's input
input_discount_rate = tk.Entry(root, width=12)
input_discount_rate.grid(row=18, column=1)



# The GUI is almost done.
# We still need to add three buttons.
# We need to define functions to tell the buttons what to do when
# they are clicked on. Thus, we will first define our functions
# and then create and place our buttons on the screen.







##################################
# Part 2: Coding the DCF Model
##################################


# In this part, we define the two functions needed for our buttons.
# Function 1 retrieves the stock price and the past growth.
# Function 2 computes the equity value per share.


# API requirements
# We import requests and pandas, they will be used
# when communicating with API. We define our API key.

import requests
import pandas as pd
key = "04871ed7089e320af811c91614b80420"


# We define those two variables outside the functions.
```

```python
# They will be used as global variables so the second function
# does need to call the API again for the same data.

cash_2020 = 0
price_of_ticker = 0







#################
# Function 1
#################


# Function number 1 takes the user's input of the ticker symbol to look up the price of the stock and
# calculate the last 4-year growth of the free cash flows.

def get_financial_data():

    #Retrieve the input of the user for the ticker symbol
    ticker = input_ticker_symbol.get()

    #Convert it in upper case for the API
    ticker = ticker.upper()

    #Getting the price of the stock
    #We found the stock price in the API under the "discounted-cash-flow" document
    document = "discounted-cash-flow"

    #Passing the parameters in the url
    url = "https://financialmodelingprep.com/api/v3/{}/{}?&apikey={}".format(document, ticker, key)
    r = requests.get(url)

    #We transform the dictionary data into a dataframe, and transpose for easier readability.
    price = pd.DataFrame.from_dict(r.json()).transpose()

    #Shift the first row as a column
    price.columns = price.iloc[0]

    #We ignore the first row
    price = price.iloc[1:]

    #Making the price_of_ticker a global variable so the second function does not have to
    #do an API call again.
    global price_of_ticker

    #We locate the price data and round it
    price_of_ticker = price.iloc[2][0]
    price_of_ticker = round(price_of_ticker, 2)

    #We create a label and position it on the screen. The price will now be displayed on the GUI.
    price_label = tk.Label(root, text=str(price_of_ticker), font=("times new roman", 14))
    price_label.grid(row=6, column=3)







    #Here we will display on the screen the last 4-yr growth of the free cash flows


    # We need to retrieve the past cash flows.
    # his data can be found under the cash flow statements of the API
    #Those are almost the same steps as above but with different parameters.
    #We get the data from the API, transpose it and convert it to a dataframe.
    document_cashflow = "cash-flow-statement"
    url_2 = "https://financialmodelingprep.com/api/v3/{}/{}?&apikey={}".format(document_cashflow, ticker, key)
    r2 = requests.get(url_2)
    cf_statement = pd.DataFrame.from_dict(r2.json()).transpose()
    cf_statement.columns = cf_statement.iloc[0]
    cf_statement = cf_statement.iloc[1:]

    #We store the cash flows in the variable cash_flows
    cash_flows = cf_statement.loc["freeCashFlow"]

    #We make the free cash flow of 2020 a global variable so the second function can
    #directly use it.
    global cash_2020
    cash_2020 = cash_flows.iloc[0]

    #Here we create a try/except since some stocks have 6 years of data instead of 5 years.
    #This is to debug the mistake "index out of bound".
    try:
        cash_2016 = cash_flows.iloc[5]
    except:
        cash_2016 = cash_flows.iloc[4]



    #Here we use a formula to compute the average growth rate of the last 4 years.
    #We use the compound annual growth rate formula
    #We convert the cash to integer for the calculation, and format the results as percentage
    average_compound_growth = (((int(cash_2020) / int(cash_2016)) ** (1 / 4)) - 1)*100

    #We round the results to two decimals place.
    average_compound_growth = round(average_compound_growth, 2)

    #We display the growth rate on the screen using label and grid. We add a "%" sign to our result.
    growth_label = tk.Label(root, text=str(average_compound_growth)+"%", font=("times new roman", 14))
    growth_label.grid(row=7, column=3)

#######################
# Button for function 1
#######################


#We use the button function of Tkinter. Under command, we precise which function to execute for our button.
#We also name our button "Get Financial data". Using grid, we position our button. We choose the size with padx and
#pady. We selected a background color with bg.
button_get_data = tk.Button(root, text="Get financial data", command=get_financial_data, padx=50, pady=8, bg="#89CFF0")
button_get_data.grid(row=7, column=0)





#################
# Function 2
#################

# Function 2 computes the intrinsic value and displays it on the screen.
# To do this, the function needs the user assumptions about growth and discount rate.
# In term of financial data, the function needs the net debt, the cash position and
# the number of shares.

def intrinsic_value():

    #We retrieve the input of the ticker symbol and put in upper case.
    #We retrieve this data by using the get function on our input label.
    ticker = input_ticker_symbol.get()
    ticker = ticker.upper()

    #We retrieve the user input regarding the growth rate.
    future_growth_rate = input_future_growth_rate.get()

    #we convert the input to int.
    future_growth_rate = int(future_growth_rate)

    #We convert the growth rate in decimals.
    future_growth_rate = future_growth_rate/100

    #Retrieve, convert in int and in decimals to input discount rate.
    discount_rate = input_discount_rate.get()
    discount_rate = int(discount_rate)
    discount_rate = discount_rate/100

    #Here we do the projections of FCF for the next 6-years.
    #This is based on the assumption of the user.
    cash_2021 = cash_2020 * (1 + future_growth_rate)
    cash_2022 = cash_2021 * (1 + future_growth_rate)
    cash_2023 = cash_2022 * (1 + future_growth_rate)
    cash_2024 = cash_2023 * (1 + future_growth_rate)
    cash_2025 = cash_2024 * (1 + future_growth_rate)
    cash_2026 = cash_2025 * (1 + future_growth_rate)

    #Here we discount each cash flow to obtain the present value.
    cash_discount_2021 = cash_2021 / (1 + discount_rate)
    cash_discount_2022 = cash_2022 / ((1 + discount_rate) ** 2)
    cash_discount_2023 = cash_2023 / ((1 + discount_rate) ** 3)
    cash_discount_2024 = cash_2024 / ((1 + discount_rate) ** 4)
    cash_discount_2025 = cash_2025 / ((1 + discount_rate) ** 5)
    cash_discount_2026 = cash_2026 / ((1 + discount_rate) ** 6)

    #We sum each discounted cash flow.
    sum_of_discounted_free_cash_flow = cash_discount_2021 + cash_discount_2022 + \
                                       cash_discount_2023 + cash_discount_2024 + cash_discount_2025 + cash_discount_2026

    #We compute the terminal value. We assume 4% long term growth rate.
    #The following lines follow the terminal value formula.
    long_term_growth_rate = 0.04
    fcf_2026 = cash_2026*(1+long_term_growth_rate)
    terminal_value = fcf_2026/(discount_rate-long_term_growth_rate)
    terminal_value_discounted = terminal_value/((1+discount_rate)**6)

    #We then add upp the terminal value with the sum of discounted FCF to get the enterprise value.
    enterprise_value = sum_of_discounted_free_cash_flow + terminal_value_discounted


    #We know we need the net debt, cash position and the number of shares to find the equity value.

    #Here we ask the API to look at the balance sheet in order to retrieve the cash position and the net debt.
    #Similar as previous steps.
    document_balance_sheet = "balance-sheet-statement"
    url_3 = "https://financialmodelingprep.com/api/v3/{}/{}?&apikey={}".format(document_balance_sheet, ticker, key)
    r3 = requests.get(url_3)
    balance_sheet = pd.DataFrame.from_dict(r3.json()).transpose()
    balance_sheet.columns = balance_sheet.iloc[0]
    balance_sheet = balance_sheet.iloc[1:]

    #We locate the cash and net debt item on the balance sheet.
    cash = balance_sheet.iloc[7][0]
    net_debt_2020 = balance_sheet.iloc[48][0]


    #Here we retrieve the number of shares from the API. The number of shares can be found in the enterprise value
    #document. The document, the ticker symbol, and our API key are passed as parameters.
    document_number_shares = "enterprise-values"
    url_4 = "https://financialmodelingprep.com/api/v3/{}/{}?limit=40&apikey={}".format(document_number_shares, ticker,
                                                                                                         key)
    r4 = requests.get(url_4)

    #Similar steps as before. Convert to dataframe. Move a row to a column.
    data_shares = pd.DataFrame.from_dict(r4.json()).transpose()
    data_shares.columns = data_shares.iloc[0]
    data_shares = data_shares.iloc[1:]

    #We more precisely find the number of shares outstanding.
    number_of_shares_outstanding = data_shares.loc["numberOfShares"]
    number_shares_2020 = number_of_shares_outstanding.iloc[0]

    #In some cases we find the number of shares to be zero due to poor documents
    # if this is the case, we take the number of shares of last year.

    if number_shares_2020 == 0:
        number_shares_2020 = number_of_shares_outstanding.iloc[1]



    #We can finally compute the equity value by following the DCF formula.
    equity_value = enterprise_value - net_debt_2020 + cash
    equity_value_per_share = equity_value / number_shares_2020

    #We round the results to two decimals.
    equity_value_per_share = round(equity_value_per_share, 2)

    #We create a label for the output to be displayed on the screen. We add a "USD" symbol for better
    #understanfding. Using grid, we positioned the element on the GUI screen.
    intrinsic_value_label = tk.Label(root, text=str(equity_value_per_share)+" USD", font=("times new roman", 14))
    intrinsic_value_label.grid(row=14, column=3)

    #We define an upside potential describing the potential gain in percentage between the calculated
    #value and the actual stock price. The upside potential is negative if the stock is overvalued.
    upside_potential = (equity_value_per_share / price_of_ticker) - 1

    #Put in percentage
    upside_potential = upside_potential*100

    #Rounding to two decimals
    upside_potential = round(upside_potential, 2)

    #Creating a label to display the upside potential as text. We convert the upside_potential into a string.
    upside_potential_label = tk.Label(root, text=str(upside_potential)+"%", font=("times new roman", 14))
    upside_potential_label.grid(row=16, column=3)



#######################
# Button for function 2
#######################

#Button for the function 2 to display the intrinsic value and the potential upside compared to the actual price.
#Using command we link the function and the button together. Padx, and pady are for the dimension of the button.
#Bg is for the background color of the button.
button_calculate_intrinsic_value = tk.Button(root, text="Calculate the intrinsic value", command=intrinsic_value,
                                             padx=50, pady=8, bg="#89CFF0")

#Postioning
button_calculate_intrinsic_value.grid(row=20, column=0)


#Blank label for spacing reasons
blank_3 = tk.Label(root, text="            ", font=("calibre", 20, "bold"))
blank_3.grid(row=19, column=0)


#Small button to quit the root GUI using the function quit.
button_quit = tk.Button(root, text="Exit Program", command=root.quit)
button_quit.grid(row=20, column=3)


#Create an infinite loop to display the GUI
root.mainloop()
```