Team 3892 Group Work Project 2 - Derivative Pricing

Question 1: **Team member A** will repeat questions 5, 6, and 7 of GWP#1 using the Black-Scholes closed-form solution to price the different European Options. For Q7 on vega, you can use Black-Scholes closed-form solution.

```
# Loading the required libaries
import numpy as np
import pandas as pd
from tqdm import tqdm
import scipy.stats as ss
from scipy.stats import lognorm, norm
import matplotlib.pyplot as plt
import math

# Input Parameters:
S0 = 100
R = 0.05
Sigma = 0.20
T = 3/12 # 3 months over 1 year

# Strike prices defined: Deep OTM, OTM, ATM, ITM, and Deep ITM (moneyness of 90%, 95%,
K_90 = 90
K_95 = 95
K_100 = 100
K_105 = 105
K_110 = 110
```

The following codes generated based on the European Options with Black-Scholes: The Black-Scholes closed-form solution provides a singular output from derived equations from GBM and Ito's Rules. The following equation is used for option pricing:

$$Option_{Call} \rightarrow c = S_0\mathcal{M}(d_1) - Ke^{-rT}\mathcal{M}(d_2)$$

$$Option_{Put} \rightarrow p = Ke^{-rT}\mathcal{M}(d_2) - S_0\mathcal{M}(d_1)$$

Where $\mathcal{M}()$ represents the CDF of the standard normal. The values are calculated using the following equations:

$$d_1 = \frac{Ln(S_0/K)+(r+\sigma^2/2)T}{\sigma\sqrt{T}}$$

$$d_2 = \frac{Ln(S_0/K)+(r-\sigma^2/2)T}{\sigma\sqrt{T}} = d_1 - \sigma\sqrt{T}$$

✓ 0s completed at 11:18 AM

```
def BS_Option_Prices(S, r, sigma, t, T, K, opt):
  ttm = T - t
  if ttm < 0:
        return 0, 0, 0, 0, 0, 0
  elif ttm == 0.0:
      return np.maximum(S - K, 0.0) if opt == "C" else np.maximum(K - S, 0.0)

  vol = sigma * np.sqrt(ttm)

  d1 = (np.log(S / K) + (r + 0.5 * vol**2) * ttm) / vol
  d2 = d1 - vol

  if opt in ["C", "P"]:
      if opt in ["C"]:
          Opt_Price = S * ss.norm.cdf(d1) - K * np.exp(-r * ttm) * ss.norm.cdf(d2)
          Delta = ss.norm.cdf(d1)
          Gamma = ss.norm.pdf(d1) / (S * sigma * np.sqrt(ttm))
          Vega = S * ss.norm.pdf(d1) * np.sqrt(ttm)
          Theta = -(S * ss.norm.pdf(d1) * sigma) / (2 * np.sqrt(ttm)) - r * K * np.exp(
              -r * ttm) * ss.norm.cdf(d2)
          Rho = K * ttm * np.exp(-r * ttm) * ss.norm.cdf(d2)
      else:
          Opt_Price = K * np.exp(-r * ttm) * ss.norm.cdf(-d2) - S * ss.norm.cdf(-d1)
          Delta = -ss.norm.cdf(-d1)
          Gamma = ss.norm.pdf(d1) / (S * sigma * np.sqrt(ttm))
          Vega = S * ss.norm.pdf(d1) * np.sqrt(ttm)
          Theta = -(S * ss.norm.pdf(d1) * sigma) / (2 * np.sqrt(ttm)) + r * K * np.exp(
              -r * ttm) * ss.norm.cdf(-d2)
          Rho = -K * ttm * np.exp(-r * ttm) * ss.norm.cdf(-d2)
  else:
      Opt_Price = "Error: option type incorrect. Choose P for a put option or C for a c
  return Opt_Price, Delta, Gamma, Vega, Theta, Rho

BS_Call = np.array(BS_Option_Prices(S0, R, Sigma, 0, T, K_100, "C"))
BS_Put = np.array(BS_Option_Prices(S0, R, Sigma, 0, T, K_100, "P"))

pd.DataFrame(
    {
      "Black-Scholes Call": [BS_Call[0],BS_Call[1]],
      "Black-Scholes Put": [BS_Put[0],BS_Put[1]],
   },
   index=["Option Price", "Greek Delta"]
).round(2)
```

This Code part is used to measure the Vega of the Option (computing the change in option price in relation to the changes in the volatility). $\nu = \frac{\partial^2 V}{\partial \sigma} = S\mathcal{N}(d_1)\sqrt{T-t}$

```
BS_HigherVolCall = np.array(BS_Option_Prices(S0, R, 0.25, 0, T, K_100, "C"))
BS_HigherVolPut = np.array(BS_Option_Prices(S0, R, 0.25, 0, T, K_100, "P"))

pd.DataFrame(
    {
      "Black-Scholes Call": [BS_Call[0],BS_HigherVolCall[0],BS_Call[3]],
      "Black-Scholes Put": [BS_Put[0],BS_HigherVolPut[0],BS_Put[3]],
   },
   index=["Base Option Price (Sigma = 0.20)","Price at Higher Volatility (Sigma = 0.25)
).round(2)
```

Team member B will repeat questions 5, 6, and 7 of GWP#1 using Monte-Carlo methods under a general GBM equation with daily time-steps in the simulations. As was the case with the number of time steps in the trees, make sure you run a large enough number of simulations. For Q7 here you can rely on the same intuition as in the trees, just 'shock' the volatility parameter and recalculate things

```
#The codes is based on Lesson note 3 of Module 4 sample codes:
def BS_MonteCarlo(S, K, r, sigma, T, t, M, Opt, Exe):
  ttm = T - t
  if ttm < 0:
        return 0, 0, 0, 0, 0, 0
  elif ttm == 0.0:
      return np.maximum(S - K, 0.0) if Opt == "C" else np.maximum(K - S, 0.0)

  data = np.zeros((M, 2))
  z = np.random.normal(0, 1, [1, M])

  ST = S * np.exp((ttm) * (r - 0.5 * sigma**2) + sigma * np.sqrt(ttm) * z)
  if Exe == "EU":
    if Opt == "C":
      data[:, 1] = ST - K
      average = np.sum(np.amax(data, axis=1)) / float(M)
      return np.exp(-r * (ttm)) * average
```

```
      return np.exp(-r * (ttm)) * average

    elif Opt == "P":
      data[:, 1] = K - ST
      average = np.sum(np.amax(data, axis=1)) / float(M)
      return np.exp(-r * (ttm)) * average


  elif Exe == "AM":
    data_early = np.zeros((M, 2))
    if Opt == "C":
      data[:, 1] = np.exp(-r * (ttm))  * (ST - K)
      exp_average = np.sum(np.amax(data, axis=1)) / float(M)
      data_early = ST - K
      average_early = np.sum(np.amax(data_early, axis=1)) / float(M)
      average = round(np.max(np.stack([exp_average,average_early]),0),2)
      return average

    elif Opt == "P":
      data[:, 1] = np.exp(-r * (ttm))  * (K - ST)
      exp_average = np.sum(np.amax(data, axis=1)) / float(M)
      data_early[:, 1] = (K - ST)
      average_early = np.sum(np.amax(data_early, axis=1)) / float(M)
      average = round(np.max(np.stack([exp_average,average_early]),0),2)
      return average


def Delta_MonteCarlo(S0, Delta_S0, K, r, sigma, T, t, M, Opt, Exe):
  PriceHi = BS_MonteCarlo(S0+Delta_S0, K_100, R, Sigma, T, t, M, Opt, Exe)
  PriceLo = BS_MonteCarlo(S0-Delta_S0, K_100, R, Sigma, T, t, M, Opt, Exe)

  return  round((PriceHi - PriceLo) / (2*Delta_S0),2)

def Vega_MonteCarlo(S0, K, r, sigma, Delta_sigma, T, t, M, Opt, Exe):
  Price1 = BS_MonteCarlo(S0, K_100, R, sigma, T, t, M, Opt, Exe)
  Price2 = BS_MonteCarlo(S0, K_100, R, sigma+Delta_sigma, T, t, M, Opt, Exe)

  return  round(abs(Price1 - Price2) / (Delta_sigma),2)

Eu_Call = BS_MonteCarlo(S0, K_100, R, Sigma, T, 0, 1000000, "C", "EU")
Eu_Call_Delta = Delta_MonteCarlo(S0, 5, K_100, R, Sigma, T, 0, 100000, "C", "EU")
Eu_Call_Vega = Vega_MonteCarlo(S0, K_100, R, Sigma, 0.05, T, 0, 1000000, "C", "EU")
Eu_Call_Higher = BS_MonteCarlo(S0, K_100, R, Sigma+0.05, T, 0, 1000000, "C", "EU")

pd.DataFrame(
    {
      "Monte Carlo European Call": [Eu_Call, Eu_Call_Delta],
   },
   index=["Option Price", "Greek Delta"]
).round(2)
```

```
Eu_Put = BS_MonteCarlo(S0, K_100, R, Sigma, T, 0, 100000, "P", "EU")
Eu_Put_Delta = Delta_MonteCarlo(S0, 5, K_100, R, Sigma, T, 0, 100000, "P", "EU")
Eu_Put_Vega = Vega_MonteCarlo(S0, K_100, R, Sigma, 0.05, T, 0, 100000, "P", "EU")
Eu_Put_Higher = BS_MonteCarlo(S0, K_100, R, Sigma+0.05, T, 0, 100000, "P", "EU")

pd.DataFrame(
    {
      "Monte Carlo European Put": [Eu_Put, Eu_Put_Delta],
   },
   index=["Option Price", "Greek Delta"]
).round(2)
```

```
M = 1000000   # Monte Carlo iterations

MC_HigherVolCall = np.array(BS_MonteCarlo(S0, K_100, R, 0.25, T, 0, M, "C", "EU"))
MC_HigherVolPut = np.array(BS_MonteCarlo(S0, K_100, R, 0.25, T, 0, M, "P", "EU"))

Eu_Call = BS_MonteCarlo(S0, K_100, R, Sigma, T, 0, M, "C", "EU")
Eu_Call_Delta = Delta_MonteCarlo(S0, 5, K_100, R, Sigma, T, 0, M, "C", "EU")
Eu_Call_Vega = Vega_MonteCarlo(S0, K_100, R, Sigma, 0.05, T, 0, M, "C", "EU")
Eu_Call_Higher = BS_MonteCarlo(S0, K_100, R, Sigma+0.05, T, 0, M, "C", "EU")

Eu_Put = BS_MonteCarlo(S0, K_100, R, Sigma, T, 0, M, "P", "EU")
Eu_Put_Delta = Delta_MonteCarlo(S0, 5, K_100, R, Sigma, T, 0, M, "P", "EU")
Eu_Put_Vega = Vega_MonteCarlo(S0, K_100, R, Sigma, 0.05, T, 0, M, "P", "EU")
Eu_Put_Higher = BS_MonteCarlo(S0, K_100, R, Sigma+0.05, T, 0, M, "P", "EU")

Ame_Call = BS_MonteCarlo(S0, K_100, R, Sigma, T, 0, M, "C", "AM")
Ame_Call_Delta = Delta_MonteCarlo(S0, 5, K_100, R, Sigma, T, 0, M, "C", "AM")
Ame_Call_Vega = Vega_MonteCarlo(S0, K_100, R, Sigma, 0.05, T, 0, M, "C", "AM")
Ame_Call_Higher = BS_MonteCarlo(S0, K_100, R, Sigma+0.05, T, 0, M, "C", "AM")

Ame_Put = BS_MonteCarlo(S0, K_100, R, Sigma, T, 0, M, "P", "AM")
Ame_Put_Delta = Delta_MonteCarlo(S0, 5, K_100, R, Sigma, T, 0, M, "P", "AM")
Ame_Put_Vega = Vega_MonteCarlo(S0, K_100, R, Sigma, 0.05, T, 0, M, "P", "AM")
Ame_Put_Higher = BS_MonteCarlo(S0, K_100, R, Sigma+0.05, T, 0, M, "P", "AM")

pd.DataFrame(
```

```
pd.DataFrame(
    {
      "Monte Carlo Amercian Call": [Ame_Call, Ame_Call_Delta],
   },
   index=["Option Price", "Greek Delta"]
).round(2)
```

```
pd.DataFrame(
    {
      "Monte Carlo Black-Scholes Call": [Ame_Call, Ame_Call_Higher, Ame_Call_Vega],
      "Monte Carlo Black-Scholes Put": [Ame_Put, Ame_Put_Higher, Ame_Put_Vega],
   },
   index=["Base Option Price (Sigma = 0.20)","Price at Higher Volatility (Sigma = 0.25)
).round(2)
```

Question 2: **Team member B** will repeat questions 5, 6, and 7 of GWP#1 using Monte-Carlo methods under a general GBM equation with daily time-steps in the simulations.

As was the case with the number of time steps in the trees, make sure you run a large enough number of simulations. For Q7 here you can rely on the same intuition as in the trees, just 'shock' the volatility parameter and recalculate things.

```
def option_payoff(S, K, optype):
  return np.maximum(S-K, 0) if optype == 'C' else np.maximum(K-S, 0)

def bs_european_option_mc(S, K, r, sigma, T, M, optype):
  # simulate the stock paths
  z = np.random.normal(0, 1, [1, M])
  ST = S * np.exp((T) * (r - 0.5 * sigma**2) + sigma * np.sqrt(T) * z)

  # stock prices at the end of the paths and the payoffs
  payoff = option_payoff(ST, K, optype)
```

```
  # the option price at time 0 is the present value of the avg option price at expirati
  option_price0 = np.exp(-r * T) * payoff

  return np.mean(option_price0)

mc_european_call_price = bs_european_option_mc(S0, K_100, R, Sigma, T, 10000000, optype
mc_european_put_price = bs_european_option_mc(S0, K_100, R, Sigma, T, 10000000, optype=

S_shock = 5
mc_european_call_price_higher = bs_european_option_mc(S0+S_shock, K_100, R, Sigma, T, 1
mc_european_put_price_higher = bs_european_option_mc(S0+S_shock, K_100, R, Sigma, T, 10
mc_european_call_price_lower = bs_european_option_mc(S0-S_shock, K_100, R, Sigma, T, 10
mc_european_put_price_lower = bs_european_option_mc(S0-S_shock, K_100, R, Sigma, T, 100

mc_european_call_delta = (mc_european_call_price_higher - mc_european_call_price_lower)
mc_european_put_delta = (mc_european_put_price_higher - mc_european_put_price_lower) /

pd.DataFrame(
    {
      "Monte Carlo Black-Scholes Call": [mc_european_call_price, mc_european_call_delta
      "Monte Carlo Black-Scholes Put": [mc_european_put_price, mc_european_put_delta],
   },
   index=["MC Option Price", "Greek Delta"]
).round(2)
```

```
sigma_shock = 0.05

# calculate the option prices, deltas and put-call parity for higher volatility
eur_call_price_mc_highersigma = bs_european_option_mc(S0, K_100, R, Sigma+sigma_shock,
eur_put_price_mc_highersigma = bs_european_option_mc(S0, K_100, R, Sigma+sigma_shock, T

# calculate vegas
print()
mc_european_call_vega = (eur_call_price_mc_highersigma-mc_european_call_price)/sigma_sh
mc_european_put_vega = (eur_put_price_mc_highersigma-mc_european_put_price)/sigma_shock

pd.DataFrame(
    {
      "Monte Carlo Black-Scholes Call": [mc_european_call_price, eur_call_price_mc_high
      "Monte Carlo Black-Scholes Put": [mc_european_put_price, eur_put_price_mc_highers
   },
   index=["Base Option Price (Sigma = 0.20)", "Price at Higher Volatility (Sigma = 0.25
).round(2)
```

Step 2: American Derivatives.

Question 4: **Team member A** will use Monte-Carlo methods with regular GBM process and daily simulations on an American Call option. Remember to answer the different questions in the original GWP#1: price (Q5), calculate delta (Q6) and vega (Q7) only for the Call option case.

```
#The codes is based on Lesson note 3 of Module 4 sample codes:
def BS_MonteCarlo(S, K, r, sigma, T, t, M, Opt, Exe):
  ttm = T - t
  if ttm < 0:
        return 0, 0, 0, 0, 0, 0
  elif ttm == 0.0:
      return np.maximum(S - K, 0.0) if Opt == "C" else np.maximum(K - S, 0.0)

  data = np.zeros((M, 2))
  z = np.random.normal(0, 1, [1, M])

  ST = S * np.exp((ttm) * (r - 0.5 * sigma**2) + sigma * np.sqrt(ttm) * z)
  if Exe == "EU":
    if Opt == "C":
      data[:, 1] = ST - K
      average = np.sum(np.amax(data, axis=1)) / float(M)
      return np.exp(-r * (ttm)) * average

    elif Opt == "P":
      data[:, 1] = K - ST
      average = np.sum(np.amax(data, axis=1)) / float(M)
      return np.exp(-r * (ttm)) * average


  elif Exe == "AM":
    data_early = np.zeros((M, 2))
    if Opt == "C":
      data[:, 1] = np.exp(-r * (ttm))  * (ST - K)
      exp_average = np.sum(np.amax(data, axis=1)) / float(M)
      data_early = ST - K
      average_early = np.sum(np.amax(data_early, axis=1)) / float(M)
      average = round(np.max(np.stack([exp_average,average_early]),0),2)
```

```
      return average

    elif Opt == "P":
      data[:, 1] = np.exp(-r * (ttm))  * (K - ST)
      exp_average = np.sum(np.amax(data, axis=1)) / float(M)
      data_early[:, 1] = (K - ST)
      average_early = np.sum(np.amax(data_early, axis=1)) / float(M)
      average = round(np.max(np.stack([exp_average,average_early]),0),2)
      return average


def Delta_MonteCarlo(S0, Delta_S0, K, r, sigma, T, t, M, Opt, Exe):
  PriceHi = BS_MonteCarlo(S0+Delta_S0, K_100, R, Sigma, T, t, M, Opt, Exe)
  PriceLo = BS_MonteCarlo(S0-Delta_S0, K_100, R, Sigma, T, t, M, Opt, Exe)

  return  round((PriceHi - PriceLo) / (2*Delta_S0),2)

def Vega_MonteCarlo(S0, K, r, sigma, Delta_sigma, T, t, M, Opt, Exe):
  Price1 = BS_MonteCarlo(S0, K_100, R, sigma, T, t, M, Opt, Exe)
  Price2 = BS_MonteCarlo(S0, K_100, R, sigma+Delta_sigma, T, t, M, Opt, Exe)

  return  round(abs(Price1 - Price2) / (Delta_sigma),2)

Ame_Call = BS_MonteCarlo(S0, K_100, R, Sigma, T, 0, 100000, "C", "AM")
Ame_Call_Delta = Delta_MonteCarlo(S0, 5, K_100, R, Sigma, T, 0, 100000, "C", "AM")
Ame_Call_Vega = Vega_MonteCarlo(S0, K_100, R, Sigma, 0.05, T, 0, 100000, "C", "AM")
Ame_Call_Higher = BS_MonteCarlo(S0, K_100, R, Sigma+0.05, T, 0, 100000, "C", "AM")

pd.DataFrame(
    {
      "Monte Carlo Amercian Call": [Ame_Call, Ame_Call_Delta],
   },
   index=["Option Price", "Greek Delta"]
).round(2)
```

```
pd.DataFrame(
    {
      "American Call": [Ame_Call, Ame_Call_Higher, Ame_Call_Vega],
   },
   index=["Base Option Price (Sigma = 0.20)","Price at Higher Volatility (Sigma = 0.25)
).round(2)
```

Question 5: **Team member B** will use Monte-Carlo methods with regular GBM process and daily simulations on an American Put option. Remember to answer the different questions in the original GWP#1: price (Q5), calculate delta (Q6) and vega (Q7) only for the **Put option case**.

```
#The codes is based on Lesson note 3 of Module 4 sample codes:
def BS_MonteCarlo(S, K, r, sigma, T, t, M, Opt, Exe):
  ttm = T - t
  if ttm < 0:
        return 0, 0, 0, 0, 0, 0
  elif ttm == 0.0:
      return np.maximum(S - K, 0.0) if Opt == "C" else np.maximum(K - S, 0.0)

  data = np.zeros((M, 2))
  z = np.random.normal(0, 1, [1, M])

  ST = S * np.exp((ttm) * (r - 0.5 * sigma**2) + sigma * np.sqrt(ttm) * z)
  if Exe == "EU":
    if Opt == "C":
      data[:, 1] = ST - K
      average = np.sum(np.amax(data, axis=1)) / float(M)
      return np.exp(-r * (ttm)) * average

    elif Opt == "P":
      data[:, 1] = K - ST
      average = np.sum(np.amax(data, axis=1)) / float(M)
      return np.exp(-r * (ttm)) * average


  elif Exe == "AM":
    data_early = np.zeros((M, 2))
    if Opt == "C":
      data[:, 1] = np.exp(-r * (ttm))  * (ST - K)
      exp_average = np.sum(np.amax(data, axis=1)) / float(M)
      data_early = ST - K
      average_early = np.sum(np.amax(data_early, axis=1)) / float(M)
      average = round(np.max(np.stack([exp_average,average_early]),0),2)
      return average

    elif Opt == "P":
      data[:, 1] = np.exp(-r * (ttm))  * (K - ST)
      exp_average = np.sum(np.amax(data, axis=1)) / float(M)
      data_early[:, 1] = (K - ST)
      average_early = np.sum(np.amax(data_early, axis=1)) / float(M)
```

```
        average = round(np.max(np.stack([exp_average,average_early]),0),2)
        return average


def Delta_MonteCarlo(S0, Delta_S0, K, r, sigma, T, t, M, Opt, Exe):
  PriceHi = BS_MonteCarlo(S0+Delta_S0, K_100, R, Sigma, T, t, M, Opt, Exe)
  PriceLo = BS_MonteCarlo(S0-Delta_S0, K_100, R, Sigma, T, t, M, Opt, Exe)

  return  round((PriceHi - PriceLo) / (2*Delta_S0),2)

def Vega_MonteCarlo(S0, K, r, sigma, Delta_sigma, T, t, M, Opt, Exe):
  Price1 = BS_MonteCarlo(S0, K_100, R, sigma, T, t, M, Opt, Exe)
  Price2 = BS_MonteCarlo(S0, K_100, R, sigma+Delta_sigma, T, t, M, Opt, Exe)

  return  round(abs(Price1 - Price2) / (Delta_sigma),2)

Ame_Put = BS_MonteCarlo(S0, K_100, R, Sigma, T, 0, 100000, "P", "AM")
Ame_Put_Delta = Delta_MonteCarlo(S0, 5, K_100, R, Sigma, T, 0, 100000, "P", "AM")
Ame_Put_Vega = Vega_MonteCarlo(S0, K_100, R, Sigma, 0.05, T, 0, 100000, "P", "AM")
Ame_Put_Higher = BS_MonteCarlo(S0, K_100, R, Sigma+0.05, T, 0, 100000, "P", "AM")

pd.DataFrame(
    {
      "Monte Carlo Amercian Put": [Ame_Put, Ame_Put_Delta],
   },
   index=["Option Price", "Greek Delta"]
).round(2)
```

```
pd.DataFrame(
    {
      "American Put": [Ame_Put, Ame_Put_Higher, Ame_Put_Vega],
   },
   index=["Base Option Price (Sigma = 0.20)","Price at Higher Volatility (Sigma = 0.25)
).round(2)
```

Step 3: Hedging under Black-Scholes Analytics for European Options

Using Pricing Difference Exotic Instruments

Question 7: **Team member A** will work with European options with same characteristics as GWP#1 under different levels of moneyness:

a. Price an European Call option with 110% moneyness and an European Put with 95% moneyness using Black-Scholes. Both have 3 months maturity.

b. You build a portfolio that buys the previous Call and Put options. What is the delta of the portfolio? How would you delta-hedge this portfolio?

c. You build a second portfolio that buys the previous Call option and sells the Put. What is the delta of the portfolio? How would you delta-hedge this portfolio?

```
# European Call with 110% Moneyness (K_110) and Put with 95% moneyness (K_95)
BS_Call_Moneyness_110 = np.array(BS_Option_Prices(S0, R, Sigma, 0, T, K_110, "C"))
BS_Put_Moneyness_95 = np.array(BS_Option_Prices(S0, R, Sigma, 0, T, K_95, "P"))

pd.DataFrame(
    {
      "European Call": [K_110,BS_Call_Moneyness_110[0],BS_Call_Moneyness_110[1]],
      "European Put": [K_95, BS_Put_Moneyness_95[0],BS_Put_Moneyness_95[1]],
   },
   index=["Strike Price", "Option Price", "Greek Delta"]
).round(2)
```

```
# Generating Delta Hedging Strategy for the Portfolio:
def Delta_Hedging(Delta_C, Number_Call, Direction_Call, Delta_P, Number_Put, Direction_
  if Direction_Call == "Buy":
    Call_Value = 100*Number_Call*Delta_C
  elif Direction_Call == "Sell":
    Call_Value = -100*Number_Call*Delta_C
  else:
    return print("Please specify the action 'Buy' or 'Sell' for the Call")
  if Direction_Put == "Buy":
```

```
    Put_Value = 100*Number_Put*Delta_P
  elif Direction_Put == "Sell":
    Put_Value = -100*Number_Put*Delta_P
  else:
    return print("Please specify the action 'Buy' or 'Sell' for the Put")

  Portfolio_Value = Call_Value + Put_Value
  if Portfolio_Value > 0:
    return print("The portfolio has a delta of {}.\nThe portfolio requires selling {} o

  elif Portfolio_Value < 0:
    return print("The portfolio has a delta of {}.\nThe portfolio requires buying {} of

  else:
    return print("The portfolio is delta neutral and requires no delta hedging.")


# 7.b: You build a portfolio that buys the previous Call and Put options. What is the d
Delta_Hedging(0.21, 1, "Buy", -0.26, 1,"Buy")
```

```
The portfolio has a delta of -0.05.
The portfolio requires buying 5.0 of the underlying stock to delta hedge.
```

```
# 7.c: ou build a second portfolio that buys the previous Call option and sells the Put
Delta_Hedging(0.21, 1, "Buy", -0.26, 1,"Sell")
```

```
The portfolio has a delta of 0.47.
The portfolio requires selling 47.0 of the underlying stock to delta hedge.
```

Question 8: **Team member B** will work with Monte-Carlo methods with daily time steps to price an Up-and-Out (UAO) barrier option. The option is currently ATM with a barrier level of 141 and: S_0 = 120; r = 6%; σ = 30%; T = 8 months

```
def option_payoff(S, K, optype):
  return np.maximum(S-K, 0) if optype == 'C' else np.maximum(K-S, 0)

def mc_bs_uao_european_option(S, K, r, sigma, T, bl, M, optype):
  N = int(np.round(T*255))
  dt = T / N

  isin = np.ones([M])

  for day in tqdm(range(N)):
    z = np.random.normal(0, 1, [1, M])
    S = S * np.exp((dt) * (r - 0.5 * sigma**2) + sigma * np.sqrt(dt) * z)

    # determining the stock price path stays below the barrier level (here 141$)
    isin = (S < bl).astype(float) * isin
```

```
    isin = (S < B1).astype(float) * isin

  payoff = option_payoff(S, K, optype) * isin
  option_price0 = np.exp(-r * T) * payoff

  return np.mean(option_price0)

mc_eur_uao_call_price = mc_bs_uao_european_option(120, 120, 0.06, 0.3, 8/12, 141, 10000

pd.DataFrame(
    {
      "Up-and-Out European Call": [mc_eur_uao_call_price],
   },
   index=["Option Price"]
).round(2)
```

```
# European OTM Put
"""
# Input Parameters:
S0 = 100
R = 0.05
Sigma = 0.20
T = 3/12 # 3 months over 1 year

# Strike prices defined: Deep OTM, OTM, ATM, ITM, and Deep ITM (moneyness of 90%, 95%,

K_95 = 95  OTM PUT


BS_Option_Prices(S, r, sigma, t, T, K, opt)

"""

European_OTM_Put = BS_Option_Prices(S0, R, Sigma, 0, T, K_95, "P")
European_OTM_Put[0].round(2)
```

```
     1.53
```

```
# American DEEP OTM Put
"""
# Input Parameters:
S0 = 100
R = 0.05
Sigma = 0.20
```

```
T = 3/12 # 3 months over 1 year

# Strike prices defined: Deep OTM, OTM, ATM, ITM, and Deep ITM (moneyness of 90%, 95%,

K_95 = 95  OTM PUT


BS_Option_Prices(S, r, sigma, t, T, K, opt)

"""

European_OTM_Put = BS_Option_Prices(S0, R, Sigma, 0, T, K_95, "P")
European_OTM_Put[0].round(2)
```

```
     1.53
```

```
# Monte Carlo American Call Price Monte Carlo Valuation with different Stikes ranging f

ame_call_MC_K_90 = BS_MonteCarlo(S=S0, K=K_90, r=R, sigma=Sigma, T=T, t=0, M=M, Opt= "C
ame_call_MC_K_95 = BS_MonteCarlo(S=S0, K=K_95, r=R, sigma=Sigma, T=T, t=0, M=M, Opt= "C
ame_call_MC_K_100 = BS_MonteCarlo(S=S0, K=K_100, r=R, sigma=Sigma, T=T, t=0, M=M, Opt=
ame_call_MC_K_105 = BS_MonteCarlo(S=S0, K=K_105, r=R, sigma=Sigma, T=T, t=0, M=M, Opt=
ame_call_MC_K_110= BS_MonteCarlo(S=S0, K=K_110, r=R, sigma=Sigma, T=T, t=0, M=M, Opt= "



pd.DataFrame(
    {
      "Monte Carlo American  Call Option Price":[X for X in [ame_call_MC_K_90,
                                                             ame_call_MC_K_95,
                                                             ame_call_MC_K_100,
                                                             ame_call_MC_K_105,
                                                             ame_call_MC_K_110]]
    },
   index = ['K = 90', 'K = 95', 'K = 100', 'K = 105', 'K = 110']).round(2)
```

```
# Monte Carlo American Put Price Monte Carlo Valuation with different Stikes ranging fr

# Monte Carlo American Put Price Valuation with different Stikes ranging from 90% tp 11
```

```
ame_put_MC_K_90 = BS_MonteCarlo(S=S0, K=K_90, r=R, sigma=Sigma, T=T, t=0, M=M, Opt= "P"
ame_put_MC_K_95 = BS_MonteCarlo(S=S0, K=K_95, r=R, sigma=Sigma, T=T, t=0, M=M, Opt= "P"
ame_put_MC_K_100 = BS_MonteCarlo(S=S0, K=K_100, r=R, sigma=Sigma, T=T, t=0, M=M, Opt= "
ame_put_MC_K_105 = BS_MonteCarlo(S=S0, K=K_105, r=R, sigma=Sigma, T=T, t=0, M=M, Opt= "
ame_put_MC_K_110= BS_MonteCarlo(S=S0, K=K_110, r=R, sigma=Sigma, T=T, t=0, M=M, Opt= "P

pd.DataFrame(
    {
      "Monte Carlo American  Put Option Price":[X for X in [ame_put_MC_K_90,
                                                            ame_put_MC_K_95,
                                                            ame_put_MC_K_100,
                                                            ame_put_MC_K_105,
                                                            ame_put_MC_K_110]]
    },
   index = ['K = 90', 'K = 95', 'K = 100', 'K = 105', 'K = 110']).round(2)
```

This is the end of GWP2 coding section!!!

Colab paid products - Cancel contracts here