Q1.

Let the following events:

H -> coin lands heads up

A -> 1^{st} coin is chosen from the bag

B -> 2^{nd} coin is chosen from the bag

C -> 3^{rd} coin is chosen from the bag

We need to compute P(A|H)

Bayes' formula: P(A|H) = $\frac{P(H|A)\times P(A)}{P(H)}$

P(A) = probability that 1^{st} coin is chosen from the bag = 1/3

Similarly, P(B) = P(C) = 1/3

P(H|A) = probability that the 1^{st} coin lands heads up = 100% = 1

Similarly, P(H|B) = 50% = ½ ; P(H|C) = 25% = 1/4

P(H) = probability that the coin lands heads up => need to use law of total probability

P(H) = P(A) × P(H|A) + P(B) × P(H|B) + P(C) × P(H|C) = 1/3 × 1 + 1/3 × ½ + 1/3 × ¼

- ⇨ P(H) = 7/12

So, P(A|H) = $\frac{P(H|A)\times P(A)}{P(H)} = \frac{1\times 1/3}{7/12} = 4/7$

Q2.

Decompose 100! = 2^a x 5^b x c

Number of trailing zeros will be min (a,b) = b – since there are fewer 5s in 100! than 2s.

In 100! we have floor(100/5) = 20 factors which are divisible by 5 (add one 5 factor in the product decomposition). We also have floor(100/25) = 4 factors which are divisible by 5^2 (add two 5 factors in the product decomposition).

So, b = floor(100/5) + floor(100/25) = 20 + 4 = 24.

Q3.

(a)

f(r) = $\sqrt{1+r}$ => f(0) = 1

$\frac{df}{dr}$ = ½ $\times$ $\frac{1}{(1+r)^{1/2}}$ => $\frac{df}{dr}(0)$= ½

1st order Maclaurin series expansion for f(r) =>

$$f(r) \approx f(0) + \frac{df}{dr}(0) \times r \Rightarrow \sqrt{1+r} \approx 1 + \frac{1}{2} \times r$$

(b)

$$\lim_{x\to\infty} \sqrt{x^2+x} - x = \lim_{x\to\infty} x\sqrt{1+\frac{1}{x}} - x = \lim_{x\to\infty} x\left[\sqrt{1+\frac{1}{x}} - 1\right] =$$

$$\lim_{x\to\infty} \frac{\sqrt{1+\frac{1}{x}}-1}{\frac{1}{x}}$$

Let $y \stackrel{\text{def}}{=} \frac{1}{x}$ => when $x \to \infty \Rightarrow y \to 0$

$$\Rightarrow \lim_{x\to\infty} \frac{\sqrt{1+\frac{1}{x}}-1}{\frac{1}{x}} = \lim_{y\to 0} \frac{\sqrt{1+y}-1}{y}$$

$$\lim_{y\to 0} \sqrt{1+y} - 1 = 0$$

$$\lim_{y\to 0} y = 0$$

Using L'Hopital rule $\left(\frac{0}{0}\right)$ $\lim_{y\to 0} \frac{\sqrt{1+y}-1}{y} = \lim_{y\to 0} \frac{\frac{\partial\sqrt{1+y}-1}{\partial y}}{\frac{\partial y}{\partial y}} =$

$$\lim_{y\to 0} \frac{1}{2}(1+y)^{-1/2} = 1/2$$

Q4.

$$\min_b E[(Y-bX)^2]$$

$$\frac{\partial E[\ldots]}{\partial b} = 0$$

2E[(Y-bX) (-X)] = 0

2E[-YX] + 2bE[XX] = 0

$$b = \frac{E[YX]}{E[XX]} = ½$$

Let c = slope of X against Y => $c = \frac{E[YX]}{E[YY]}$ (similar calculation as above)

$Var[X] = E[X^2] – E^2[X] = 1$ & $E[X] = 0 \Rightarrow E[X^2] = 1$

$E[XX] = 1$ & $\frac{E[YX]}{E[XX]} = ½ \Rightarrow E[YX] = ½$

$Var[Y] = E[Y^2] – E^2[Y] = 1$ & $E[Y] = 0 \Rightarrow E[Y^2] = 1$

$c = \frac{E[YX]}{E[YY]} = \frac{1/2}{1} = ½$

Q5.

(a) print(x[-1])

(b)

```
x[0] = 1
x[1] = 1
for i = 3; i<=100; i++
        x[i] = x[i-1] + x[i-2]
```

(c) Polymorphism is one of the 4 pillars of OOP. Through polymorphism, the methods which have the same signature can exhibit different behaviour in different classes (which are all linked together by inheritance). E.g. we have 3 classes: option, call_option and put_option. The implementation of the calculate_payoff() method below is an example of polymorphism:

```
class option:
        def calculate_payoff(S, K):
                pass


class call_option(option):

        def calculate_payoff(S,K):

                return max(0, S-K)



class put_option(option):

        def calculate_payoff(S,K):

                return max(0, K-S)
```

Q6.

Let

P = price of bond

CF_t = bond cash flow at time t

y = YTM

T = bond maturity/term

C = convexity of bond

⇨ $C = \frac{1}{P \times (1+y)^2} \times \sum_{t=1}^{T} [\frac{CF_t}{(1+y)^t} \times (t^2 + t)]$

Since interest rates at all tenors are currently 0% => y = 0%

⇨ $C = \frac{1}{P} \times \sum_{t=1}^{T} [CF_t \times (t^2 + t)]$

Let $\tau \in [1, T]\ fixed$

⇨ $\frac{\partial C}{\partial CF_\tau} = \frac{1}{P} \times (\tau^2 + \tau) > 0 =>$ need to fix the value of the CFs (otherwise, convexity increases w/ CF).

Assume $\sum CF_t = X\ fixed$

$\frac{\partial C}{\partial \tau} = \frac{1}{P} \times CF_\tau \times (2 \times \tau + 1) > 0 => \tau \in [1, T]\ and\ C \uparrow \tau =>$ max C when τ = T => the more CF we allocate towards maturity, the higher the convexity of the bond => allocate all CF at maturity => get a zero-coupon bond.

Optimal cash flow profile = $\begin{pmatrix} 0 \\ 0 \\ \cdots \\ X \end{pmatrix}$

Q7.

Let Var(A) = variance(A) => volatility(A) = $\sqrt{Var(A)}$ = V => Var(A) = V^2

Similarly Var(B) = V^2

Let Cov(X,Y) = covariance between X & Y

Let Corr(X,Y) = correlation between X & Y

Var(aX+bY) = a^2 Var(X) + b^2 Var(Y) + 2abCov(X,Y)

⇨ Var(A-B) = Var(A) + Var(B) – 2Cov(A,B)

- ⇨ $\frac{V}{2} = V^2 + V^2 - 2Cov(A,B)$
- ⇨ $Cov(A,B) = V^2 - \frac{V}{4}$

Corr(A,B) = $\frac{Cov(A,B)}{\sqrt{Var(A) \times Var(B)}}$ => Corr(A,B) = $\frac{V^2 - \frac{V}{4}}{V \times V} = 1 - \frac{1}{4 \times V}$

Q8.

Let a portfolio Π = Long underlying + Long put option (strike Kd) + Short call option (strike Ku). This portfolio will have the same payoff as the main option (see below proof). By arbitrage, the main option will be equivalent to portfolio Π.

Π = S + P(Kd) – C(Ku)

Delta of main option = Delta of Portfolio

- ⇨ Delta of Portfolio = Delta of S + Delta of P(Kd) – Delta of C(Ku)

Delta of S = $\frac{\partial S}{\partial S}$ = 1

Delta of P(Kd) = $\frac{\partial P(Kd)}{\partial S} = N(d1) - 1$

Delta of C(Ku) = $\frac{\partial C(Ku)}{\partial S} = N(d1)$

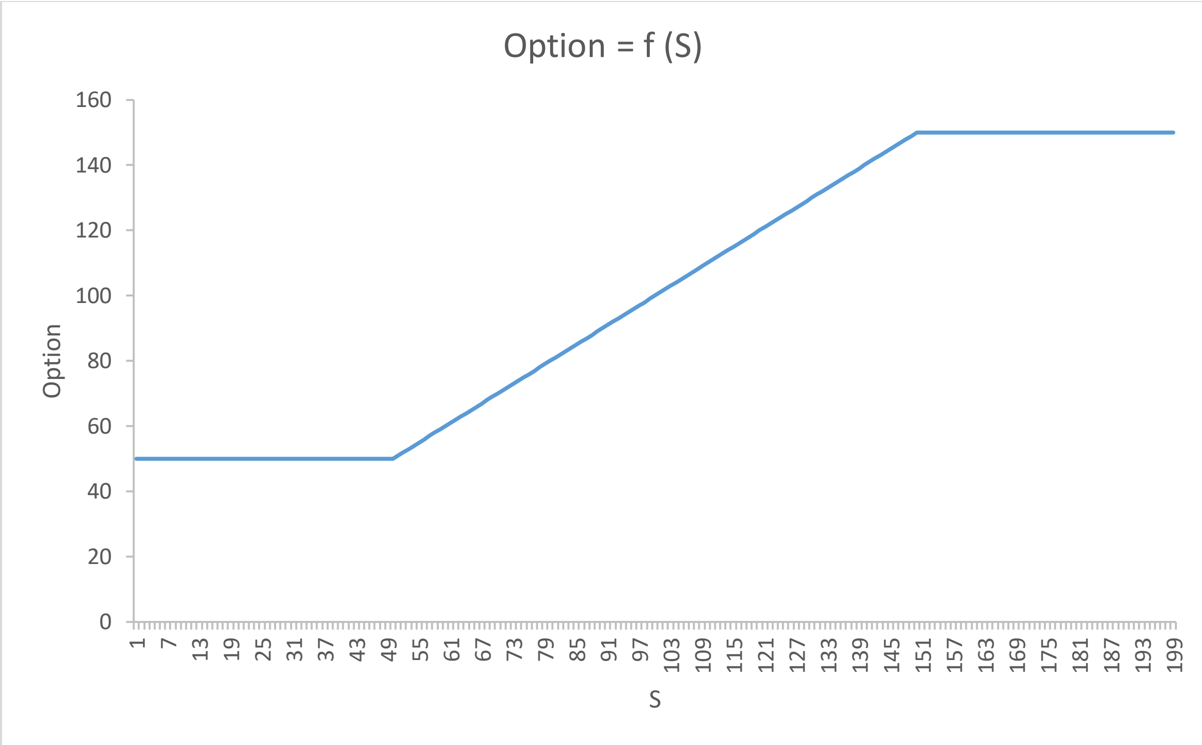
- ⇨ **Delta of Portfolio = 1 + N(d1) – 1 – N(d1) = 0**

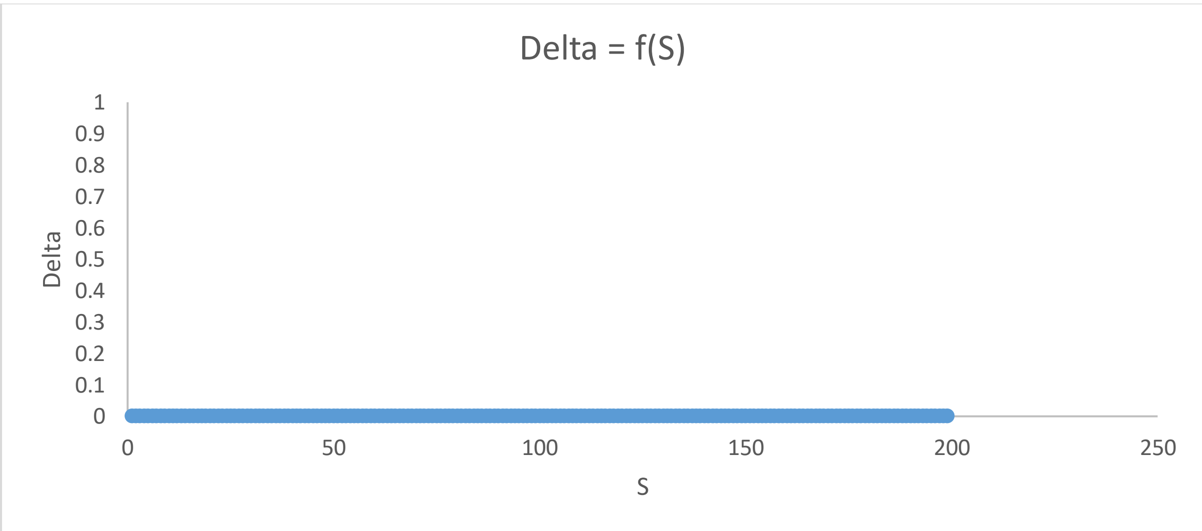
Gamma of main Option = Gamma of Portfolio

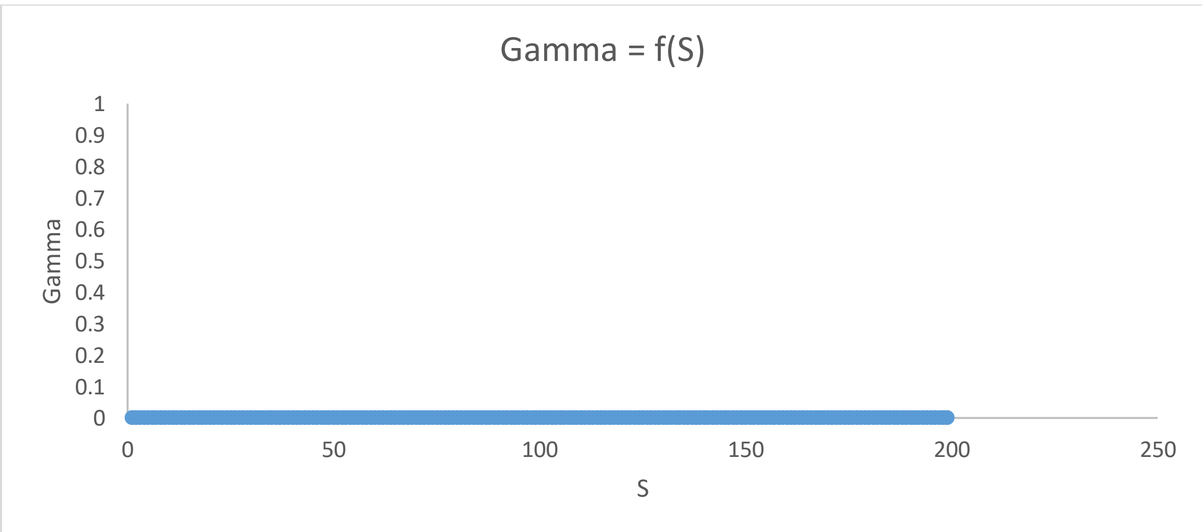
- ⇨ **Gamma of Portfolio =** $\frac{\partial \boldsymbol{Delta\ of\ Portfolio}}{\partial \boldsymbol{S}}$ **= 0**

	S	0	Kd		Ku	+Inf
L	S	S	Kd	S	Ku	S
L	P(Kd)	Kd - S	0	0	0	0
S	C(Ku)	0	0	0	0	Ku - S
Portofolio		Kd	Kd	S	Ku	Ku

Graphs below:


Option = f (S)
Option
160
140
120
100
80
60
40
20
0
1 7 13 19 25 31 37 43 49 55 61 67 73 79 85 91 97 103 109 115 121 127 133 139 145 151 157 163 169 175 181 187 193 199
S


Delta = f(S)
Delta
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0 50 100 150 200 250
S


Gamma = f(S)
Gamma
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0 50 100 150 200 250
S

Q9.

a) Want high correlation: need to construct one of the series as a linear transformation of the other.
Want weak statistical significance of correlation: need to create 2 time series with small sample sizes (e.g. n = 4).

Python code:

```
import random as rd

from scipy.stats.stats import pearsonr

n=4

t = range(1,n)

x = [rd.random() for m in t]

y = [2*m + rd.random() for m in x]

print("Correlation coefficient: " + str(pearsonr(x,y)[0]))

print("p-value of correlation coefficient: " + str(pearsonr(x,y)[1]))
```

Results:

```
Correlation coefficient: 0.8902377967351918
p-value of correlation coefficient: 0.30107618297766015
```

b) Want low & positive correlation: use Choleski decomposition of desired correlation matrix to construct the series based on a predefined correlation coefficient. Start off with 2 uncorrelated series X =[X_1, X_2]. Compute the Choleski decomposition based on the target correlation matrix C = L * L^T. Compute Y = [Y_1, Y_2] = L * X. Y_1 & Y_2 will have desired correlation coefficient.
Want strong statistical significance of correlation: need to create 2 time series with large sample sizes (e.g. n = 4000).

Python code:

```
import random as rd

from scipy.stats.stats import pearsonr

import math

import scipy as sp
```

```
import numpy as np

n=4000

corr = 0.1

cov_matrix = np.array([[1,corr],[corr,1]])


cholesky_decomp_lower = sp.linalg.cholesky(cov_matrix, lower=True)

uncorrelated = np.random.standard_normal((2, n))

correlated = np.dot(cholesky_decomp_lower, uncorrelated)

X, Y = correlated

print("Correlation coefficient: " + str(pearsonr(X,Y)[0]))

print("p-value of correlation coefficient: " + str(pearsonr(X,Y)[1]))
```

Results:

```
Correlation coefficient: 0.10054141739215479
p-value of correlation coefficient: 1.8540986646081107e-10
```

Q10.

a+b+c = 1

$a \in S \text{ and } |S| = 21$ => a can take 21 values

Fix a = 100% => b + c = 0% => $\{b, c\} \in \{(0\%, 0\%)\}$ (1 case)

Fix a = 95% => b + c = 5% => $\{b, c\} \in \{(0\%, 5\%), (5\%, 0\%)\}$ (2 cases)

Fix a = 90% => b + c = 10% => $\{b, c\} \in \{(0\%, 10\%), (5\%, 5\%), (10\%, 0\%)\}$ (3 cases)

Fix a = 85% => b + c = 15% => $\{b, c\} \in \{(0\%, 15\%), (5\%, 10\%), (10\%, 5\%), (15\%, 0\%)\}$ (4 cases)

By now we should see a pattern. Reducing a by 5% increases the number of (b,c) pairs by 1. Just one more for testing:

Fix a = 80% => b + c = 20% => $\{b, c\} \in$
$\{(0\%, 20\%), (5\%, 15\%), (10\%, 10\%), (15\%, 5\%), (20\%, 0\%)\}$ (5 cases) q.e.d

…

Fix a = 0% => b + c = 100% => $\{b, c\} \in \{(0\%, 100\%), (5\%, 95\%), \ldots, (100\%, 0\%)\}$ (21 cases)

Total number of elements in the desired set: 1 + 2 + 3 + … + 21 = 231

Q11.

See code solution under LGIM Islands.

Requirements: Python 3.7+, PyCharm 2018.2.1+, numpy 1.19.1, pytest 6.0.1

Instructions:

1) Change the value of x in Main.py/ReadInputs()
2) Run Main.py
3) (Optional) Run UnitTests.py