6G3Z3107 Coursework

Q2: Compute the following: (a) 2/5 + 1/4(2/7 – 1/3), accurately;

(b) cos (π/5) – sin (π/3), to five decimal places.

In [1]:
```
# 2(a)
from fractions import Fraction
Fraction(2 , 5) + Fraction(1 , 4) * (Fraction(2 , 7) - Fraction(1 , 3))
```

Out[1]:
```
Fraction(163, 420)
```

In [2]:
```
# 2(b)
from math import * # Import all functions from the math library
round(cos(pi / 5) - sin(pi / 3) , 5)
```

Out[2]:
```
-0.05701
```

Q3: The logisitic map function, f say, is defined as: $f(x)$ = 3.5x(1 – x) .

Write a Python program that defines this function and use a loop to compute the first 20 iterates of: xn+1 = $f(xn$), given that x0 = 0.1. Evaluate to 10 decimal places throughout and put the sequence of values in a list.

In [3]:
```
def logistic_map(x):
    return 3.5 * x * (1 - x)

x0 = 0.1

values = [x0]

# Compute the first 20 iterates
for _ in range(20):
    x_next = logistic_map(values[-1])
    # Round to 10 decimal places and append to the list
    values.append(round(x_next, 10))

# Print the sequence of values
print(values)
```

```
[0.1, 0.315, 0.7552125, 0.6470330295, 0.7993345088, 0.5613959814, 0.861806
8671, 0.4168352682, 0.8507926958, 0.444305696, 0.8641435058, 0.4108982751,
0.8472130892, 0.4530507474, 0.8672851869, 0.4028555702, 0.8419703592, 0.46
5696957, 0.8708815543, 0.3935640544, 0.8353498632]
```

Q4: Use the turtle module to plot the figure:

Use a notebook for this question and use lists of lists. The red “F” is plotted using a list of lists with eleven points (vertices). The blue “F” is the red “F” rotated by 3π/4 radians, clockwise.

You must use the rotation matrix:

(cos (θ) –sin (θ)

sin (θ) cos(θ)), to rotate the image.

In [4]:
```
!pip install ColabTurtlePlus
from ColabTurtlePlus.Turtle import *
```

```
Collecting ColabTurtlePlus
  Downloading ColabTurtlePlus-2.0.1-py3-none-any.whl (31 kB)
Installing collected packages: ColabTurtlePlus
Successfully installed ColabTurtlePlus-2.0.1
Put clearscreen() as the first line in a cell (after the import command) t
o re-run turtle commands in the cell
```

In [5]:
```
import turtle
import math

# Set up the screen
screen = turtle.Screen()
screen.setup(width=800, height=600)

# Set up the turtle
t = turtle.Turtle()
t.speed(0)  # Fastest speed

# Define the red "F" points (list of lists)
red_F_points = [
    [0, 0], [0, 100], [50, 100], [50, 80],
    [20, 80], [20, 60], [40, 60], [40, 40],
    [20, 40], [20, 0], [0, 0]]

# Function to draw the shape
def draw_shape(points, color):
    t.penup()
    t.goto(points[0][0], points[0][1])
    t.pendown()
    t.color(color)
    t.begin_fill()
    for point in points:
        t.goto(point[0], point[1])
    t.end_fill()

# Draw the red "F"
draw_shape(red_F_points, 'red')

# Function to rotate points
def rotate_points(points, angle):
    rotated_points = []
    cos_theta = math.cos(angle)
    sin_theta = math.sin(angle)
    for x, y in points:
        x_new = x * cos_theta + y * sin_theta
        y_new = -x * sin_theta + y * cos_theta
        rotated_points.append([x_new, y_new])
    return rotated_points

# Rotate the red "F" by 3π/4 radians clockwise
angle = 3 * math.pi / 4
blue_F_points = rotate_points(red_F_points, angle)
```

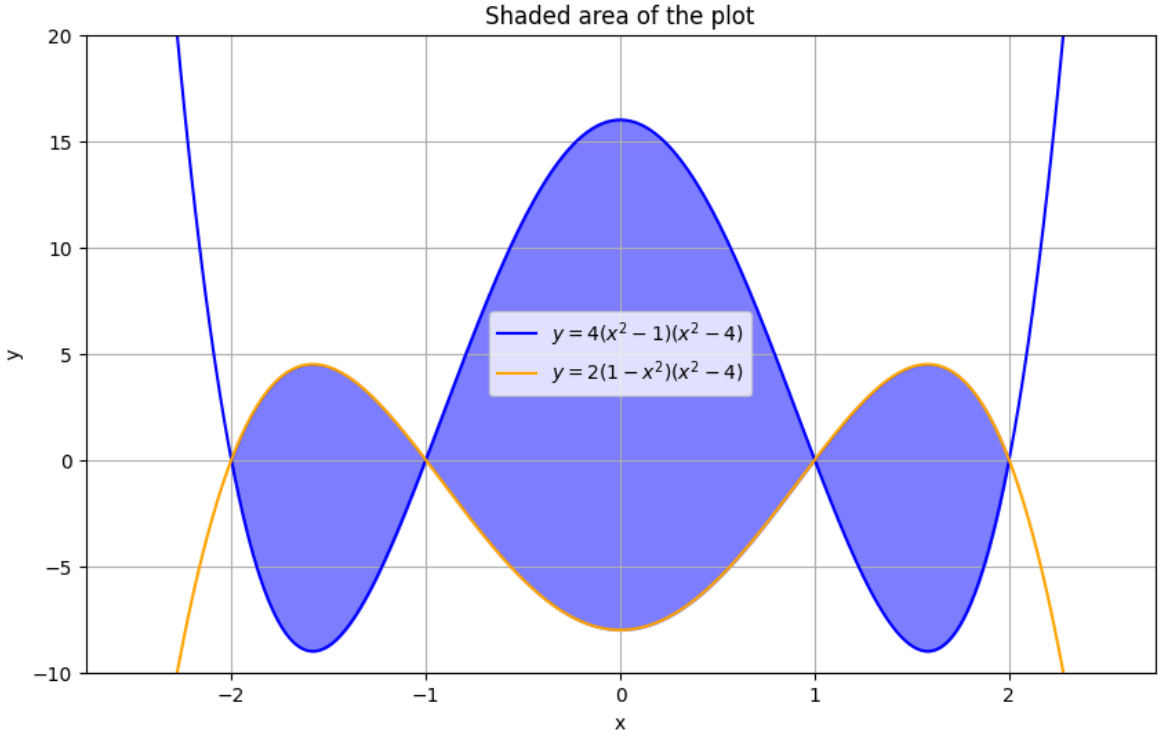
```
# Draw the blue "F"
draw_shape(blue_F_points, 'blue')

# Hide the turtle and finish
t.hideturtle()
turtle.done()
```

```
---------------------------------------------------------------------------
_
TclError                                  Traceback (most recent call las
t)
<ipython-input-5-afcd1ae38360> in <cell line: 5>()
      3
      4 # Set up the screen
----> 5 screen = turtle.Screen()
      6 screen.setup(width=800, height=600)
      7

/usr/lib/python3.10/turtle.py in Screen()
   3662     else return the existing one."""
   3663     if Turtle._screen is None:
-> 3664         Turtle._screen = _Screen()
   3665     return Turtle._screen
   3666

/usr/lib/python3.10/turtle.py in __init__(self)
   3678         # preserved (perhaps by passing it as an optional paramete
r)
   3679         if _Screen._root is None:
-> 3680             _Screen._root = self._root = _Root()
   3681             self._root.title(_Screen._title)
   3682             self._root.ondestroy(self._destroy)

/usr/lib/python3.10/turtle.py in __init__(self)
    433     """Root class for Screen based on Tkinter."""
    434     def __init__(self):
--> 435         TK.Tk.__init__(self)
    436
    437     def setupcanvas(self, width, height, cwidth, cheight):

/usr/lib/python3.10/tkinter/__init__.py in __init__(self, screenName, base
Name, className, useTk, sync, use)
   2297                 baseName = baseName + ext
   2298         interactive = False
-> 2299         self.tk = _tkinter.create(screenName, baseName, className,
interactive, wantobjects, useTk, sync, use)
   2300         if useTk:
   2301             self._loadtk()

TclError: no display name and no $DISPLAY environment variable
```

Q5: Use matploblib to plot the figure and determine the shaded area

In [6]:
```
import numpy as np
import matplotlib.pyplot as plt

# Define the functions
def f1(x):
    return 4 * (x**2 - 1) * (x**2 - 4)
```

```
def f2(x):
    return 2 * (1 - x**2) * (x**2 - 4)

# Define the range for x
x = np.linspace(-2.5, 2.5, 400)
y1 = f1(x)
y2 = f2(x)

# Plot the functions
plt.figure(figsize=(10, 6))
plt.plot(x, y1, label=r'$y = 4(x^2 - 1)(x^2 - 4)$', color='blue')
plt.plot(x, y2, label=r'$y = 2(1 - x^2)(x^2 - 4)$', color='orange')

# These intervals are identified based on the graph's intersection points
plt.fill_between(x, y1, y2, where=((x >= -2) & (x <= -1)), color='blue',
plt.fill_between(x, y1, y2, where=((x >= -1) & (x <= 1)), color='blue', a
plt.fill_between(x, y1, y2, where=((x >= 1) & (x <= 2)), color='blue', al

plt.legend()
plt.xlabel('x')
plt.ylabel('y')
plt.ylim(-10, 20)
plt.title('Shaded area of the plot')
plt.grid(True)
plt.show()
```

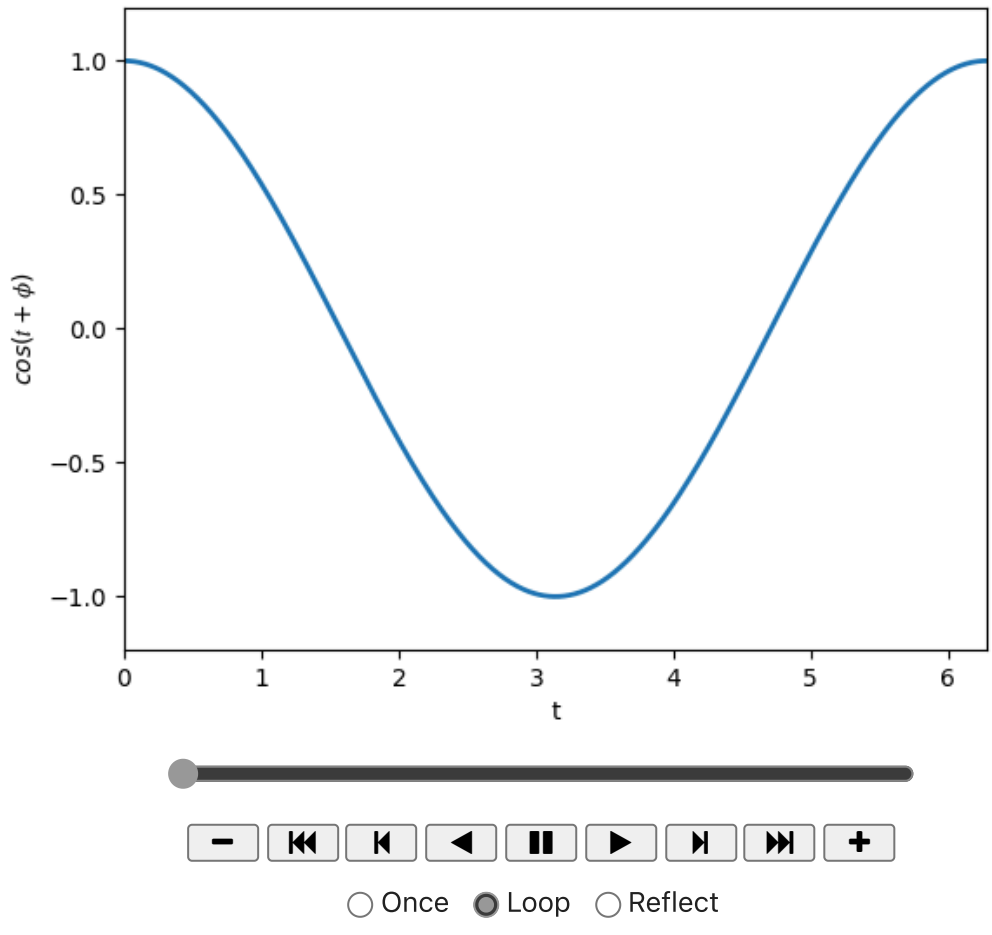


Q6: The Python program for animating the curve $y = \sin(\omega t)$, for $0 \le \omega \le 5$, is posted on Moodle (Animation.ipynb saved as Animation.pdf). Edit this program to produce an animation of the curve: $y = \cos(t + \phi)$, $0 \le \phi \le \pi$. Include comments in your program on how the animation works

In [7]:
```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation

def cos_function(t, phi):
```

```
    return np.cos(t + phi)

fig, ax = plt.subplots()
ax.set_xlim(0, 2 * np.pi)
ax.set_ylim(-1.2, 1.2)
line, = ax.plot([], [], lw=2)
plt.xlabel('t')
plt.ylabel('$cos(t + \phi)$')
plt.close()

def init():
    line.set_data([], [])
    return line,

# The function to animate.
def animate(phi):
    t = np.linspace(0, 2 * np.pi, 1000)
    y = cos_function(t, phi)
    line.set_data(t, y)
    return line,

# Create the animation
phi_values = np.linspace(0, np.pi, 100)
anim = FuncAnimation(fig, animate, init_func=init, frames=phi_values, int

from IPython.display import HTML
HTML(anim.to_jshtml())
```

Out[7]:



Comments: The animation works as I changed the function to cosine in which cos_function calculates the cosine value for given t and phi. The 'animate' function updates the plot for each value of π by recalculating the cosine values and updating the plot data.

Q7: Research the Playfair Cipher on the Web. Use your first name and surname as the encryption/decryption key (without repetition of letters) and complete a 5 × 5 matrix table, missing out the letter J.

Explain how the cipher works, given that the message is “I LOVE PYTHON.” Determine the encrypted message by hand. DO NOT WRITE A PYTHON PROGRAM FOR THIS QUESTION.

$$\text{Matrix} = \begin{pmatrix} M & E & L & I & S \\ A & C & H & U & B \\ D & F & G & K & N \\ O & P & Q & R & T \\ V & W & X & Y & Z \end{pmatrix}.$$

First, split the message into digraphs. If there's an odd number of letters, add an "X" to the end:

"I LOVE PYTHON" -> "IL", "OV", "EP", "YT", "HO", "N".

Now, encrypt each digraph:

IL -> "IK" (since they are in different rows and columns, use rectangle rule)

OV -> "VD" (same row, replace with letters to the right)

EP -> "IG" (same row, replace with letters to the right)

YT -> "VU" (same column, replace with letters below)

HO -> "ND" (same row, replace with letters to the right)

N -> "NX" (adding 'X' as padding) -> "ND" (same row, replace with letters to the right)

The final encrypted message is: IKVDIGVUNDND

Q8: Given that $x1$ = 0.5, $b1$ = –0.2, $b2$ = 0.4, $w1$ = 0.2, $w2$ = 0.8, yt = 0.4, the learning rate, η = 0.1, and the activation function, $\sigma(v)$ =1/(1+e^−v) using the artificial neural network (ANN) below ($b1$, $b2$ remain constant):

(a) use Python to compute the output of this ANN, y = σ($o1$);

(b) use Python to update the weights $w1$ and $w2$ after back-propagation.

In [8]:
```
import math
```

```
# 8(a)
# Given values
x1 = 0.5
b1 = -0.2
b2 = 0.4
w1 = 0.2
w2 = 0.8
yt = 0.4
eta = 0.1

def sigmoid(v):
    return 1 / (1 + math.exp(-v))

# Forward pass
h1 = x1 * w1 + b1
sigma_h1 = sigmoid(h1)

o1 = sigma_h1 * w2 + b2
y = sigmoid(o1)

print(f"Output y: {y}")

# Backpropagation
Err = 0.5 * (yt - y) ** 2

# Gradients
dErr_dy = y - yt
dy_do1 = y * (1 - y)
do1_dw2 = sigma_h1
do1_db2 = 1

# 8(b)
# Update weights w2
w2 = w2 - eta * dErr_dy * dy_do1 * do1_dw2
# Update weights w1
dh1_dw1 = x1
dy_dh1 = w2 * y * (1 - y)
w1 = w1 - eta * dErr_dy * dy_do1 * dy_dh1 * dh1_dw1

print(f"Updated w1: {w1}")
print(f"Updated w2: {w2}")
```

```
Output y: 0.6856837023869975
Updated w1: 0.19947114544622715
Updated w2: 0.7970752494517895
```

Q9: Use the Data_1_AQA.xlsx large data set for this question.

(a) Plot graphs to compare Regions and Life Expectancies at Birth in 1960 compared to 2000. What can you conclude? What do you think the results will be like in 2040?

(b) For each Region, see how Unemployment compares with GDP per Capita (US$). What can you conclude?

In [16]:
```
import pandas as pd # Import pandas for data analysis
import seaborn as sns # Import seaborn for visualisations
import matplotlib.pyplot as plt
df1 = pd.read_excel("Data-1-AQA.xlsx" , sheet_name = "Data")
df1.iloc[0 : 237 , [0, 2, 3, 14, 18]]
```

Out[16]:

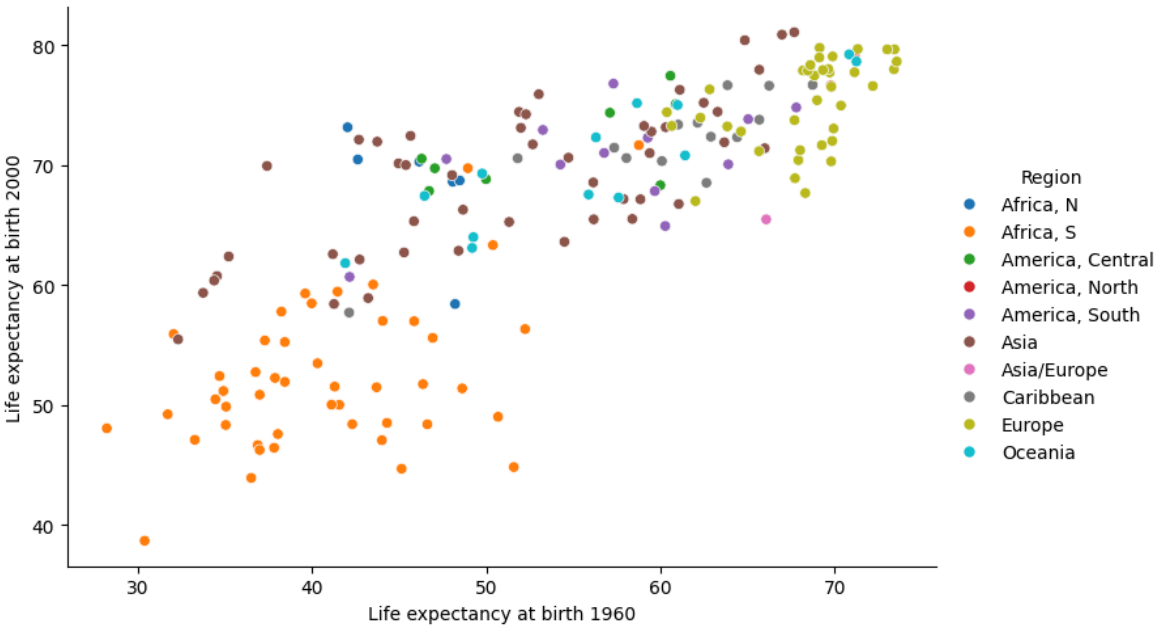
	no	Region	population	Life expectancy at birth 1960	Life expectancy at birth 2000
0	1	Africa, N	41657488	46.138	70.292
1	2	Africa, N	99413317	48.056	68.613
2	3	Africa, N	6754507	42.609	70.473
3	4	Africa, N	34314130	48.458	68.722
4	5	Africa, N	43120843	48.194	58.430
...	...	...	...	...	...
231	232	Oceania	1285	NaN	NaN
232	233	Oceania	106398	61.401	70.811
233	234	Oceania	11147	NaN	NaN
234	235	Oceania	288037	46.441	67.438
235	236	Oceania	15763	NaN	NaN

236 rows × 5 columns

In [17]:
```
# 9(a)
fig = plt.figure()
sns.relplot(data=df1, x="Life expectancy at birth 1960", \
 y="Life expectancy at birth 2000", \
 hue="Region", aspect=1.5)
```

Out[17]: <seaborn.axisgrid.FacetGrid at 0x7b79050819c0>

<Figure size 640x480 with 0 Axes>



There is a clear positive correlation between life expectancy at birth in 1960 and life expectancy at birth in 2000. Regions with higher life expectancy in 1960 tend to have higher life expectancy in 2000 as well. South Africa is the region with the lowest life expectancy overall while Europe have the highest life expectancy.

Given the trends observed from 1960 to 2000, it is reasonable to predict the following for life expectancy at birth by 2040 continues to improve. Most regions are likely to see continued increases in life expectancy due to ongoing advancements in medical technology, healthcare access, and living standards.

In [18]:
```
import pandas as pd # Import pandas for data analysis
import seaborn as sns # Import seaborn for visualisations
import matplotlib.pyplot as plt
df1 = pd.read_excel("Data-1-AQA.xlsx" , sheet_name = "Data")
df1.iloc[0 : 237 , [0, 2, 3, 8, 9]]
```

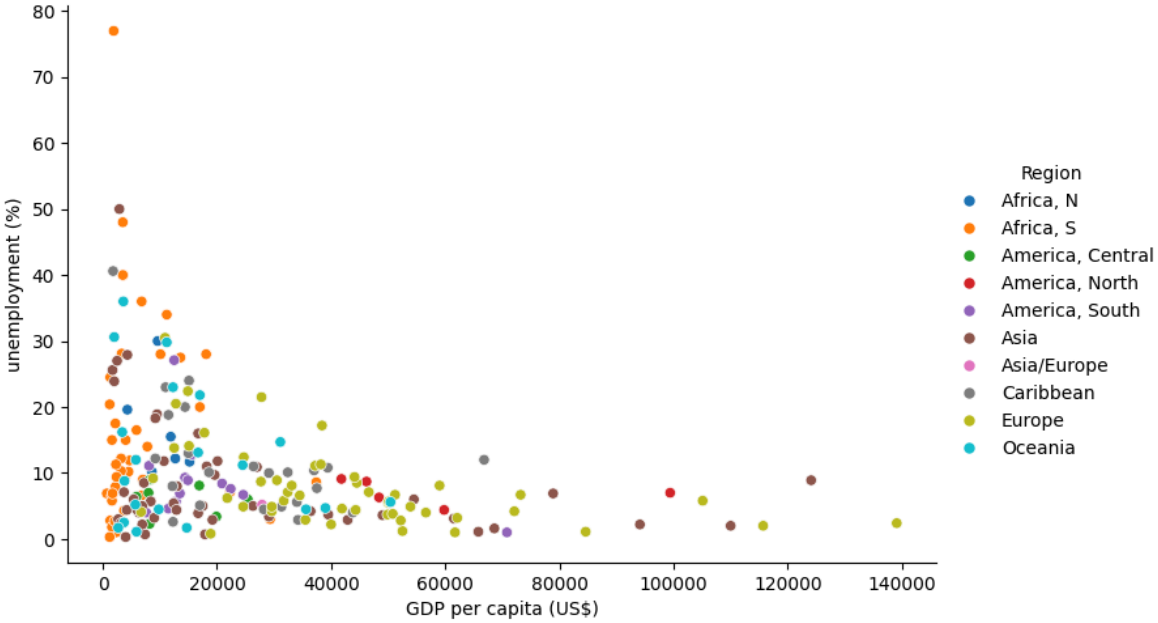
Out[18]:

	no	Region	population	unemployment (%)	GDP per capita (US$)
0	1	Africa, N	41657488	11.7	15200.0
1	2	Africa, N	99413317	12.2	12700.0
2	3	Africa, N	6754507	30.0	9600.0
3	4	Africa, N	34314130	10.2	8600.0
4	5	Africa, N	43120843	19.6	4300.0
...	...	...	...	...	...
231	232	Oceania	1285	NaN	1000.0
232	233	Oceania	106398	1.1	5900.0
233	234	Oceania	11147	NaN	3800.0
234	235	Oceania	288037	1.7	2700.0
235	236	Oceania	15763	8.8	3800.0

236 rows × 5 columns

In [19]:
```
# 9(b)
fig = plt.figure()
sns.relplot(data=df1, x="GDP per capita (US$)", \
 y="unemployment (%)", \
 hue="Region", aspect=1.5)
```

Out[19]:
```
<seaborn.axisgrid.FacetGrid at 0x7b7904ea4910>
<Figure size 640x480 with 0 Axes>
```



There is a general trend indicating a negative correlation between GDP per capita and unemployment rates. As GDP per capita increases, the unemployment rate tends to decrease.

The graph suggests that countries with higher GDP per capita are more likely to have lower unemployment rates, highlighting the impact of economic development on employment. Wealthier nations tend to offer more job opportunities and have more robust economic structures.

Q10: The following Python program shows a parent class “Pet” and child classes “Cat” and “Canary.” The objects Tom and Cuckoo have been declared.

What output do the following lines give?

print(Tom.legs)

print(Tom.tail)

print(Cuckoo.legs)

Tom.meeow()

Cuckoo.chirp()

In [20]:
```
class Pet:
  def __init__(self, legs):
    self.legs = legs

  def walk(self):
    print("Pet parent class. Walking...")

class Cat(Pet):
  def __init__(self, legs, tail):
    self.legs = legs
    self.tail = tail
```

```
  def meeow(self):
    print("Cat child class. A cat meeows but a canary can't. Meeow...")

class Canary(Pet):
  def chirp(self):
    print("Canary child class. A canary chirps but a cat can't. Chirp..."

Tom = Cat(4, True)
Cuckoo = Canary(2)
```

In [21]:
```
print(Tom.legs)
```

```
4
```

In [22]:
```
print(Tom.tail)
```

```
True
```

In [23]:
```
print(Cuckoo.legs)
```

```
2
```

In [24]:
```
Tom.meeow()
```

```
Cat child class. A cat meeows but a canary can't. Meeow...
```

In [25]:
```
Cuckoo.chirp()
```

```
Canary child class. A canary chirps but a cat can't. Chirp...
```