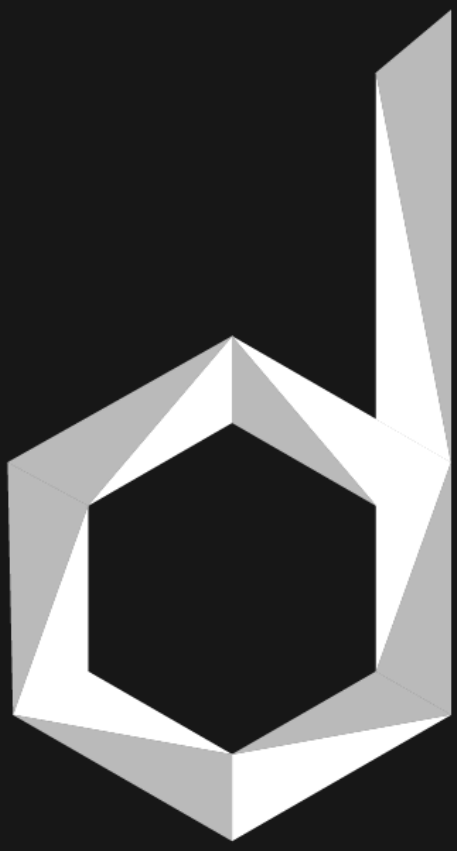
INGENIERÍA MECATRÓNICA



DI_CERO

DIEGO CERVANTES RODRÍGUEZ

DATA SCIENCE

PYTHON 3.9.7, C# & LABVIEW

Ejercicios de P00

Contenido

Teoría – POO (Programación Orientada a Objetos):

El término de clase es uno de los pilares fundamentales de la programación orientada a objetos (POO), OPP por su acrónimo en inglés.

La POO provee una aproximación para estructurar programas y aplicaciones de tal forma que sus datos y operaciones se agrupen en clases para que se puedan acceder a ellos por medio de objetos, de esta forma en un solo tipo de dato se incluyen ya acciones, características o valores.

- Por ejemplo, la clase animales se puede usar en un programa desarrollado con un estilo de programación POO, donde cada animal tiene su nombre, vida promedio, distribución geográfica, especie, etc. Además, de que pueden realizar acciones diferentes como son: comer, producir sonidos, etc. Los datos de esta clase hipotética son nombre, ubicación geográfica, especie, etc. Mientras que las operaciones que se le pueden asociar a estos datos pueden ser moverse(), comer(), producir_sonido(), etc.

Las clases actúan entonces como modelos o moldes que se utilizan para construir varias instancias o ejemplos de objetos similares. Una instancia o un objeto es un ejemplo de una clase. Todas las instancias/objetos de una clase poseen el mismo tipo de variables de datos (características) y acciones que pueden realizar.

- Tomando como ejemplo a la clase animales, sus características en común serían por enunciar algunas: su nombre, vida promedio y especie. Aunque para cada animal los datos específicos son diferentes, es decir, no existen dos animales de distinto tipo que tengan el mismo nombre, ni la misma vida promedio y ni la especie.
- Por otro lado, las operaciones que se pueden hacer con los datos son comunes para todos los objetos de una clase, aunque cada una de estas operaciones sobre un objeto específico, regresa un valor específico para este objeto.

Las clases les permiten a los programadores almacenar de forma conceptual información relacionada, lo cual hace que los códigos sean más fáciles de estructurar y de mantener.

La terminología de las clases es:

- **Clase:** Define la combinación de datos y comportamientos que operan sobre estos. Una clase actúa como un modelo o molde a través del cual se pueden crear nuevas instancias.
- **Instancias u objetos:** Una instancia también conocida como objeto, es un ejemplo de una clase. Todas las instancias de una clase poseen los mismos datos: *campos (fields)/atributos (attributes)*; pero un valor propio para cada uno de estos. Cada instancia de una clase responde al mismo conjunto de acciones: *peticiones (requests)*.
- **Atributo/Campo/Variable de instancia:** Los datos o variables que posee un objeto se representan por sus atributos. El estado de un objeto en un momento particular se relaciona con los valores corrientes que poseen sus atributos.
- **Método/Función:** Acción que pueden realizar todas las instancias de una clase y se define dentro del paréntesis que describe el parámetro del método al crear un objeto.
- **Mensaje/Parámetro:** Éste se envía a un objeto para solicitar una operación que se realizará en algún atributo de la clase que se va a acceder, solicitando al objeto hacer y/o retornar algo.

La programación orientada a objetos presenta algunas ventajas y algunas desventajas.

Dentro de las ventajas se pueden enunciar las siguientes:

- Se mejora la productividad al desarrollar software, debido a que esta programación es modular.
- Se mejora el mantenimiento del software.
- Se desarrollan el software más rápido.
- El costo del desarrollo de software es menor.
- El software suele ser de mayor calidad.

Algunas desventajas que tiene la OOP son:

- Presenta una curva de aprendizaje más compleja y en ocasiones más prolongada.
- Los programas suelen ser más largos, es decir, tienden a requerir más líneas de código.
- Los programas suelen ser más lentos que los que se desarrollan con el paradigma de la programación estructurada.
- No es adecuada para todos los tipos de problemas de programación.

Ejercicios – POO:

1.- Clase Y: Cinemática (Movimiento Rectilíneo Uniformemente Acelerado)

Escriba una clase en Pyhton y en C# llamada Y para calcular la función de cinemática que corresponde a una caída libre dada por:

$$f(t) = y_0 + v_0(t) + \frac{g(t^2)}{2}$$

Donde y_0 es la posición inicial de la partícula, v_0 es la velocidad inicial, t es el tiempo, y g es la aceleración de la gravedad dada por 9.81 m/s2.

Desarrolle la clase considerando lo siguiente:

1. El constructor de la clase debe tener como argumentos a y_0 y v_0; así como tiene que establecer el valor de g como 9.81.
2. Se debe crear un método especial llamado __call__ en Python o de un método llamado Value en C# para evaluar la función $f(t)$ cuando se proporciona un valor de t.
3. Se creará un método en la clase Y llamado formula que imprima en consola la fórmula de la función como $f(t) = y_0 + v_0(t) + \frac{g(t^2)}{2}$ con los valores de y_0 y v_0 que ingresó el usuario.
4. Graficar y_0 vs t en la misma figura con una gráfica simple matplotlib y una pyplot.

Código Python – Visual Studio Code (Logo Azul): Declaración y uso de la clase Y

```
# -*- coding: utf-8 -*-


#Comentario de una sola linea con el simbolo #, en Python para nada se deben poner acentos sino el programa
#puede fallar o imprimir raro en consola, la siguiente línea de código es para que no tenga error, pero aún
```

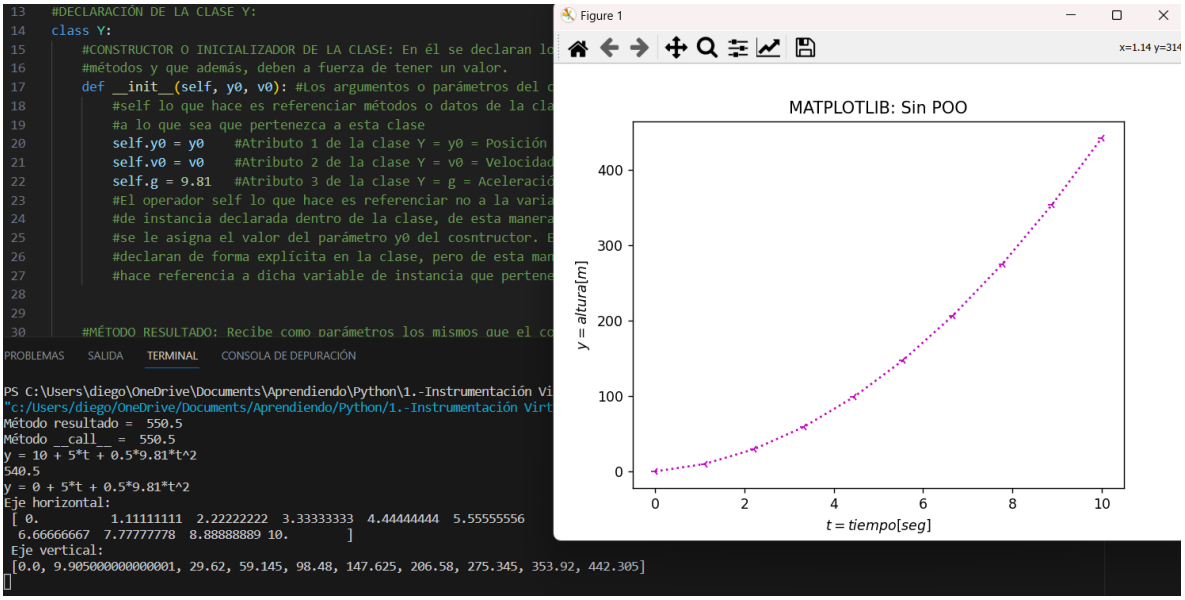
```
#así al poner un ángulo saldrá raro en consola, la línea debe ponerse tal cual como aparece y justo al inicio.

#POO: La programación orientada a objetos es una forma de estructurar programas y aplicaciones de tal forma que
#los datos y las operaciones con estos se agrupan en clases para que se puedan acceder a estos por medio de
#objetos.
#1.-Y: Clase que utiliza la fórmula cinemática de Movimiento Rectilíneo Uniformemente Acelerado para calcular
#la posición de una partícula.

#DECLARACIÓN DE LA CLASE Y:
class Y:
    #CONSTRUCTOR O INICIALIZADOR DE LA CLASE: En él se declaran los atributos que se reutilizarán en los demás
    #métodos y que además, deben a fuerza de tener un valor.
    def __init__(self, y0, v0): #Los argumentos o parámetros del constructor de la clase Y son y0 y v0.
        #self lo que hace es referenciar métodos o datos de la clase donde nos encontramos, es una referencia
        #a lo que sea que pertenezca a esta clase
        self.y0 = y0    #Atributo 1 de la clase Y = y0 = Posición inicial (Altura inicial)
        self.v0 = v0    #Atributo 2 de la clase Y = v0 = Velocidad inicial
        self.g = 9.81   #Atributo 3 de la clase Y = g = Aceleración de gravedad [m/s^2]
        #El operador self lo que hace es referenciar no a la variable local del constructor, sino a la variable
        #de instancia declarada dentro de la clase, de esta manera indicamos que a la variable de instancia y0
        #se le asigna el valor del parámetro y0 del cosntructor. En python las variables de instancia no se
        #declaran de forma explícita en la clase, pero de esta manera se indica que el parámetro de la función
        #hace referencia a dicha variable de instancia que pertenece a esta clase en específico.


    #MÉTODO RESULTADO: Recibe como parámetros los mismos que el constructor (y0 y v0), pero además la variable t.
    #Evaluación de la fórmula a través de un método llamado resultado, este hace lo mismo que el método __call__
    #lo que tiene de especial este método es la forma en la que se manda a llamar fuera de la clase, pero en sí
    #podemos hacer que realice la operación que sea y solo puede existir un método __call__ por cada clase.
    def resultado(self, t):
        #Fórmula de la Caída Libre (Movimiento Rectilíneo Uniformemente Acelerado o MRUA):
        #y = y0 + v0*t +1/2(g*t^2); El exponente en Python se denota poniendo dos signos de multiplicación; ^ = **
        #y = Altura, y0 = Altura inicial
        #v0 = Velocidad inicial
        #t = Tiempo
        #g = Gravedad
        return self.y0 + self.v0*t + 0.5*self.g*t**2


    #MÉTODO CALL: Este método es propio del lenguaje de programación Python y para utilizarze se declarar el nombre
    #del objeto y sus parámetros, a diferencia de los demás donde se declara el nombre del objeto más el nombre del
    #método:
    # - ObjetoClase(parámetro1, parámetro2, ..., parámetro_n)                    Método call
    # - ObjetoClase.NombreMétodo(parámetro1, parámetro2, ..., parámetro_n)       Método cualquiera
    #Solo puede existir un método call por clase y este trabaja con los atributos del constructor y puede incluir
```

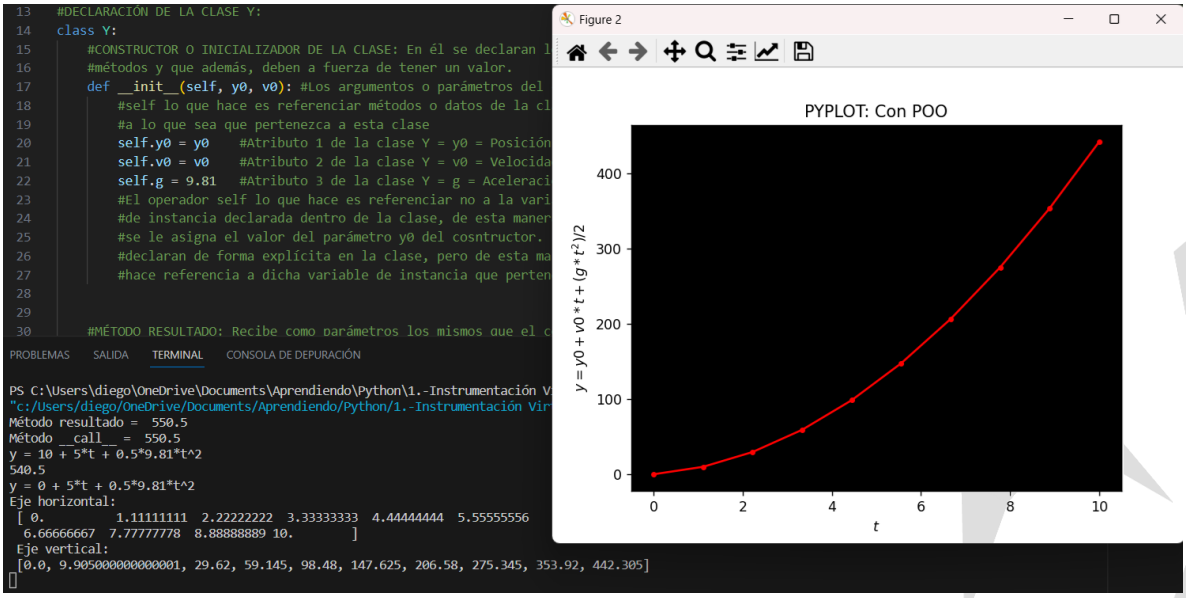
```
    #otros.
    def __call__(self, t):
        return self.y0 + self.v0*t + 0.5*self.g*t**2


    #MÉTODO FÓRMULA: Método de la clase para imprimir en pantalla el resultado de la operación.
    def formula(self):
        #print(): Imprimir un mensaje en consola.
        print("y = %g + %g*t + 0.5*%g*t^2"%(self.y0, self.v0, self.g))



#USO DE LA CLASE Y:
#Operaciones con la clase y, creando un objeto que la instancíe.
y0 = 10
v0 = 5
objeto_y = Y(y0, v0)    #Creación del objeto o instancia de la clase Y
t = 10                  #Evaluar en el tiempo t = 100 segundos en el método resultado
#Utilización del método llamado resultado perteneciente a la clase Y para resolver la ecuación de MRUA: Realiza
#la operación de la ecuación de Movimiento Rectilíneo Uniformemente Acelerado (MRUA).
y_resultado = objeto_y.resultado(t)
print("Método resultado = ", y_resultado)
#Utilización del método call de la clase Y, como es especial no se debe indicar su nombre: Realiza la operación.
y_call = objeto_y(t)
print("Método __call__ = ", y_call)
#Utilización del método formula de la clase Y: Este sirve para imprimir en consola el resultado de la operación.
objeto_y.formula()
#De esta manera se puede acceder al atributo y0 del constructor perteneciente a la clase Y:
objeto_y.y0 = 0
print(objeto_y(t))
objeto_y.formula()



#GRÁFICACIÓN DEL MOVIMIENTO RECTILÍNEO UNIFORMEMENTE ACELERADO:
import matplotlib.pyplot as plt #matplotlib: Librería de graficación matemática.
import numpy as np #Librería numpy: Realiza operaciones matemáticas complejas (matriciales) y manejar datos.

#Lista vacía para rellenarla con los valores de la distancia y para cada tiempo de t.
y_graph = []
#Bucle for que va de 0 a t-1, ya que el valor de t nunca lo toca
for i in range (0, t):
    #append(): Lo que hace es agregar un elemento a la lista (array de python).
    #Para obtener el resultado de la altura se debe usar una instancia de la clase que use el metodo resultado().
    y_graph.append(objeto_y.resultado(i))
```

```
#numpy.linspace(): Método que rellena de números un vector, indicando su número de inicio, número final y numero
#de datos.
t_graph = np.linspace(0, t, t) #Vector que va del número 0 al 15 y se compone de 15 datos.
print("Eje horizontal:\n", t_graph, "\n", "Eje vertical:\n", y_graph)



#MATPLOTLIB: Graficación sin un estilo de Programación Orientada a Objetos
#matplotlib.figure(): Método para ordenar los figures (ventanas) donde se muestran las diferentes gráficas.
plt.figure(1)
#matplotlib.plot(): Método usado para crear la gráfica, indicando como primer parámetro su eje horizontal, luego
#su eje vertical y finalmente el estilo de la gráfica.
# - Colores:                C1: color naranja, r: color rojo, b: color azul, g: verde, c: cyan, m: morado,
#   y: amarillo, k: negro, w: blanco.
# - Tipo de marcadores:     o: círculos, +: símbolos de más, .; puntos, v: Triángulo hacia abajo, h: Hexágono,
#   s: cuadrados, 1: tri abajo, 2: tri arriba, 3: tri izquierda, 4: tri derecha, etc.
# - Tipo de Líneas:         -: sólida, --: punteada (líneas), :: punteada (puntos), -.: línea y punto, 'or': Nada.
#numpy.sin(x): Con la librería numpy se pueden crear los datos de amplitud de la gráfica del seno por medio de un
#vector x que indique los datos del eje horizontal.
plt.plot(t_graph, y_graph, 'm3:') #'m3:' significa m: color magenta, 3: simbolo de tri izquierda, :: línea punteada
#matplotlib.title(): Título del figure.
plt.title(r'MATPLOTLIB: Sin POO')
#matplotlib.xlabel(): Texto que aparece en el eje horizontal.
plt.xlabel(r'$t = tiempo [seg]$')
#matplotlib.ylabel(): Texto que aparece en el eje vertical.
plt.ylabel(r'$y = altura [m]$')
#matplotlib.show(): Método para mostrar la gráfica creada.
plt.show()



#PYPLOT: Graficación con un estilo de Programación Orientada a Objetos, se crea el objeto ax de la clase plt.
fig = plt.figure(2)
#matplotlib.axes(): Método orientado a objetos usado para crear la rejilla en la ventana de graficación.
ax = plt.axes()
#axes.plot(): Método usado para crear la gráfica en la rejilla previamente creada en la ventana de graficación,
#indicando como primer parámetro su eje horizontal, luego su eje vertical y finalmente el estilo de la gráfica:
# - Colores:                C1: color naranja, r: color rojo, b: color azul, g: verde, c: cyan, m: morado,
#   y: amarillo, k: negro, w: blanco.
# - Tipo de marcadores:     o: círculos, +: símbolos de más, .; puntos, v: Triángulo hacia abajo, h: Hexágono, etc.
# - Tipo de Líneas:         -: sólida, --: punteada (líneas), :: punteada (puntos), -.: línea y punto, 'or': Nada.
ax.plot(t_graph, y_graph, 'r.-') #'r.-' significa r: color rojo, .: simbolo de punto, -: línea sólida
#axes.set_title(): Título del figure.
ax.set_title(r'PYPLOT: Con POO')
```

```
#axes.set_xlabel(): Método para indicar el texto que aparece en el eje x.
ax.set_xlabel(r'$t$')
#axes.set_ylabel(): Método para indicar el texto que aparece en el eje y.
ax.set_ylabel(r'$y = y0 + v0*t +(g*t^2)/2$')
#axes.set_facecolor(): Método para indicar el color de fondo de la gráfica, el cual puede ser indicado por los
#mismos colores previamente mencionados en el método plot() o se pueden usar los siguientes con el código xkcd:
# - Colores: xkcd:aqua, xkcd:aquamarine, xkcd:azure, xkcd:beige, etc. Los colores se pueden obtener de este link:
#https://matplotlib.org/stable/tutorials/colors/colors.html
ax.set_facecolor('xkcd:black')
#matplotlib.show(): Método para mostrar la gráfica creada.
plt.show()
```

Resultado del Código Python





Código C# (.NET Framework) – Visual Studio (Logo Morado)

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace h_ProgramacionOrientada_aObjetos
{
    //Declaración de la clase Y
    class Y
    {
        //Variables de instancia, son los datos que manejará la clase
        public double y0;
        public double v0;
        public double g;

        //Constructor de la clase
        public Y(double y0, double v0)
        {
            /*El operador this lo que hace es referenciar no a la variable local del constructor, sino a la variable
            de instancia declarada dentro de la clase, de esta manera indicamos que a la variable de instancia y0
            se le asigna el valor del parámetro y0 del cosntructor.*/
            this.y0 = y0;
            this.v0 = v0;
            g = 9.81;
        }

        //Métodos cualquiera de la clase, ya declarado el constructor no se debe usar el operador this
        public double Valorcito(double t)
        {
            return y0 + v0 * t + 0.5 * g * Math.Pow(t, 2);
        }
        public void Formulita()
        {
            Console.WriteLine("y = {0} + {1}*t + 0.5*{2}*t^2", y0, v0, g);
        }
    }//Clase Y

    class POO
    {
        static void Main(string[] args)
        {
            double y0 = 10;
            double v0 = 5;
            double t = 15;

            Y objeto_y = new Y(y0, v0); //Creación de un objeto de la clase Y
            double y_eval = objeto_y.Valorcito(t);
            Console.WriteLine("y(t = {0} = {1})", t, y_eval);
            objeto_y.Formulita();

            objeto_y.y0 = 0;
            double yConY0Cero = objeto_y.Valorcito(t);
            Console.WriteLine("y(t = {0} = {1})", t, yConY0Cero);
            objeto_y.Formulita();
        }//Método main
    }//Clase default del proyecto
}//Espacio de nombres
```

Resultado del Código C#

2.- Clase Account: Manejo de Balance de una Cuenta

Desarrolle una clase que maneje la cuenta de una persona.

Desarrolle la clase considerando lo siguiente:

1. El constructor de la clase debe tener como argumentos el nombre (name), el número de cuenta (account_number) y un monto inicial (initial_amount).
2. Se debe crear un método llamado deposit para realizar un depósito.
3. Se debe crear un método llamado withdraw para realizar un retiro.
4. A través de un método llamado dump se imprimirá en consola el nombre de la persona y el balance (estado) de la cuenta.

Código Python – Visual Studio Code (Logo Azul): Declaración de la clase Account

```
# -*- coding: utf-8 -*-

#Comentario de una sola linea con el simbolo #, en Python para nada se deben poner acentos sino el programa
#puede fallar o imprimir raro en consola, la siguiente línea de código es para que no tenga error, pero aún
#así al poner un ángulo saldrá raro en consola, la línea debe ponerse tal cual como aparece y justo al inicio.

#POO: La programación orientada a objetos es una forma de estructurar programas y aplicaciones de tal forma que
#los datos y las operaciones con estos se agrupan en clases para que se puedan acceder a estos por medio de
#objetos.
#2.-Account: Clase que sirve para realizar depósitos, retiros y balances de cuenta de una persona.

#DECLARACIÓN DE LA CLASE ACCOUNT:
class Account:
    #CONSTRUCTOR O INICIALIZADOR DE LA CLASE: En él se declaran los atributos que se reutilizarán en los demás
    #métodos y que además, deben a fuerza de tener un valor.
    def __init__(self, name, account_number, initial_amount):
    #Los parámetros del constructor de la clase Account son el nombre, número de cuenta y cantidad inicial.
        #self es distinto a this, por eso no debe tener el mismo nombre que el parámetro que está recibiendo el
        #constructor, se puede renombrar la variable que llega a la clase por medio del constructor.
        #self lo que hace es referenciar métodos o datos de la clase donde nos encontramos, es una referencia
        #a lo que sea que pertenezca a esta clase.
        self.name = name                #Atributo 1 de la clase Account = name = Nombre
        self.no = account_number        #Atributo 2 de la clase Account = account_number = número de cuenta
        self.balance = initial_amount   #Atributo 3 de la clase Account = initial_amount = cantidad inicial

    #MÉTODO DEPOSIT: Recibe como parámetros los mismos que el constructor (name, account_number e initial_amount),
    #pero además la variable amount. Sirve para hacer un deposito en la cuenta.
    def deposit(self, amount):
        self.balance = self.balance + amount

    #MÉTODO WITHDRAW: Recibe como parámetros los mismos que el constructor (name, account_number e initial_amount),
```

```
    #pero además la variable amount. Sirve para hacer un retiro de la cuenta.
    def withdraw(self, amount):
        self.balance = self.balance - amount

    #MÉTODO DUMP: Recibe como parámetros los mismos que el constructor (name, account_number e initial_amount).
    #Sirve para imprimir en consola el estado (balance) de la cuenta.
    def dump(self):
        s = "%s, %s, balance: %s" %(self.name, self.no, self.balance)
        #print(): Imprimir un mensaje en consola.
        print(s)
#Fin de la clase Acount
```

Código Python – Visual Studio Code (Logo Azul): Uso de la clase Account

```
# -*- coding: utf-8 -*-

#Comentario de una sola linea con el simbolo #, en Python para nada se deben poner acentos sino el programa
#puede fallar o imprimir raro en consola, la siguiente línea de código es para que no tenga error, pero aún
#así al poner un ángulo saldrá raro en consola, la línea debe ponerse tal cual como aparece y justo al inicio.

#Importa de la clase Acount del archivo que originalmente era 11.-POO_Cuenta y se cambió porque Python no
#puede manejar bien este nombre de archivo para importar

#USO DE LA CLASE ACCOUNT: Cuando se quiera importar una clase, el nombre de esta no puede empezar con un número,
#sino cuando la quiera importar obtendré un error y se va accediendo a las carpetas o también llamados paquetes
#en la programación orientada a objetos (POO), por medio de puntos:
# - Directorio normal:      carpeta1/carpeta2/carpeta3
# - Directorio paquetes:    carpeta1.carpeta2.carpeta3
#La parte del directorio se coloca después de la palabra reservada from y la clase a importar después de import.
from Archivos_Ejercicios_Python.POO_17_2_Account.POO_Account_Declaración import Account

#Objetos o Instancias de la clase Account: Reciben los parámetros del constructor de la clase Account, que son
#el nombre, número de cuenta y cantidad inicial de la cuenta.
a1 = Account("Helmer Homero", "1234", 20000)
a2 = Account("Pita amor", "5678", 2000)

#Métodos: Para utilizar los métodos de una clase se debe usar la siguiente nomenclatura:
# - nombreObjeto.nombreMétodo(parámetro1, parámetro2, ..., parámetro_n).
#MÉTODO DEPOSIT: Sirve para hacer un deposito en la cuenta, osea sumar a su cantidad inicial o actual.
a1.deposit(1000)        #Depósito en la cuenta del Objeto 1 = 20000 + 1000 = 21000

#MÉTODO WITHDRAW: Sirve para hacer un retiro de la cuenta, osea restar a su cantidad inicial o actual.
a1.withdraw(4000)       #Retiro en la cuenta del Objeto 1 = 21000 - 4000 = 17000
a2.withdraw(10500)      #Retiro en la cuenta del Objeto 2 = 2000 - 10500 = -8500
a1.withdraw(3500)       #Retiro en la cuenta del Objeto 1 = 17000 - 3500 = 13500
```

```python

#MÉTODO DUMP: Sirve para imprimir en consola el estado (balance) de la cuenta.
a1.dump()               #Balance de la cuenta del Objeto 1 = 13500
a2.dump()               #Balance de la cuenta del Objeto 2 = -8500

#ENCAPSULAMIENTO: Todos los atributos (variables de la clase) están encapsulados, esto significa que no pueden
#ser accedidos desde fuera de la clase, esto solamente puede ser burlado cuando existe un método que sirva para
#editar dicho valor o cuando se ponga el nombre del atributo sin un doble guión antes de su nombre.
#Se puede acceder directamente a los atributos a través de un doble guión bajo seguido del nombre de la variable:
# - nombreObjeto.__nombreAtributo: Atributo encapsulado.
# - nombreObjeto.nombreAtributo: Atributo sin encapsular.
a2.__balance = 10000    #Esto no se podrá hacer porque está encapsulado el atributo (variable) la clase
a2.dump()
a2.balance = 10000      #Si se quita el doble guión en la clase Account si se podrá cambiar el valor
a2.dump()
```

Resultado del Código Python

```
PROBLEMAS   SALIDA   TERMINAL   CONSOLA DE DEPURACIÓN

PS C:\Users\diego\OneDrive\Documents\Aprendiendo\Python\1.-Instrumentación Virtual> & D:/Users/diego/AppData/Local/Programs/Python/Python39/python.exe
"c:/Users/diego/OneDrive/Documents/Aprendiendo/Python/1.-Instrumentación Virtual/17.2.-POO Cuenta Uso.py"
Helmer Homero, 1234, balance: 13500
Pita amor, 5678, balance: -8500
Pita amor, 5678, balance: -8500
Pita amor, 5678, balance: 10000
PS C:\Users\diego\OneDrive\Documents\Aprendiendo\Python\1.-Instrumentación Virtual>
```

Código C# (.NET Framework) – Visual Studio (Logo Morado)

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace i_Ejercicio12POO_2
{
    //Clase nueva creada
    class Account
    {
        //Readonly es para encapsular el valor de la variable y que solo se pueda modificar desde dentro de la clase
        private readonly string name;
        private readonly string account_number;
        private double balance;

        //Constructor de la clase
        public Account(string name, string account_number, double initalAmount)
        {
            this.name = name;
            this.account_number = account_number;
            balance = initalAmount;
        }
        //Método depositar
        public void Deposit(double amount)
        {
            balance = balance + amount;
        }
        //Método retirar
        public void Withdraw(double amount)
        {
            balance = balance - amount;
        }
        //Método imprimir en consola el nombre, número de cuenta y crédito de la cuenta
```

```
        public void Dump()
        {
            Console.WriteLine("{0}, {1}, balance: {2}", name, account_number, balance);
        }
    }//Clase cuenta

    //Clase default que se crea al hacer un nuevo proyecto y contiene el método main
    class ClaseDefault
    {
        static void Main(string[] args)
        {
            //Instancia de la clase Account
            Account a1 = new Account("Helmer Homero", "1234", 20000);
            Account a2 = new Account("Pita amor", "5678", 2000);

            //Utilización de los métodos de la clase con cada objeto
            a1.Deposit(1000);
            a1.Withdraw(4000);
            a2.Withdraw(10500);
            a1.Withdraw(3500);

            a1.Dump();
            a2.Dump();
            //Haciendo trampa: Si esto no lo comento el programa me dará error porque está intentando acceder a una
            //variable encapsulada
            //a2.balance = 10000;
            //a2.dump();
        }
    }
}
```

Resultado del Código C#

3.- Clase Polynomial: Suma, Resta, Multiplicación, Derivación y Evaluación de Polinomios

Escriba una clase en Pyhton llamada Polynomial para sumar, restar, multiplicar, derivar y evaluar polinomios, mostrando su correcta representación en consola.

Desarrolle la clase considerando lo siguiente:

1. El constructor de la clase debe tener como argumento a los coeficientes del polinomio en una lista, o un arreglo de enteros, en Python o en C#, respectivamente. Nótese que un polinomio se puede escribir como:

   $$p(x) = a_0x^0 + a_1x^1 + \cdots + a_ix^i = a_0 + a_1x + \cdots + a_ix^i$$

   Donde los coeficientes del polinomio son: $a_0, a_1, \ldots, a_i$.

   Por ejemplo, el polinomio $p(x) = 1 - x^2 + 2x^3$, tiene como coeficientes la lista $[1,\ 0,\ 1,\ 2]$. El constructor de la función tiene que inicializar la variable de instancia coeficientes.

2. Creación de un método especial llamado __call__ en Python o Eval, en C#; para evaluar $p(x)$ cuando se proporciona un valor de x. La evaluación del polinomio requiere calcular la sumatoria:

$$p(x) = \sum_{i=0}^{n} a_i x^i$$

3. Creación de un método especial llamado __add__, que es una sobrecarga del operador +, en Python o en C#; respectivamente. Sume con este método dos polinomios, considere que los polinomios no necesariamente tienen el mismo grado.
4. Creación de un método especial llamado __sub__, que es una sobrecarga del operador -, en Python o en C#; respectivamente. Reste con este método dos polinomios, considere que los polinomios no necesariamente tienen el mismo grado.
5. Creación de un método especial llamado __mul__, que es una sobrecarga del operador *, en Python o en C#; respectivamente. Multiplique con este método los siguientes polinomios p y q: $p(x) = \sum_{i=0}^{M} c_i x^i$ y $q(x) = \sum_{j=0}^{N} d_j x^i$, donde el producto obtenido es descrito con la siguiente fórmula:

$$\left(p(x) = \sum_{i=0}^{M} c_i x^i\right) * \left(q(x) = \sum_{j=0}^{N} d_j x^i\right) = \sum_{i=0}^{M} \sum_{j=0}^{N} c_i * d_j * x^{i+j}$$

   Donde la doble sumatoria se obtiene por medio de dos ciclos for. Inicialmente, se tiene que crear una lista con un número de elementos M + N + 1, debido a que el grado más alto del polinomio es M + N y un término adicional debido al término constante del polinomio.

6. Creación de un método llamado derivative para diferenciar un polinomio. La derivada se puede realizar por medio de la formula, donde la derivada resultante tiene n−1 elementos.:

$$\frac{d}{dx}\left(\sum_{i=0}^{n} c_i x^i\right) = \frac{d\left(\sum_{i=0}^{n} c_i x^i\right)}{dx} = \sum_{i=1}^{M} i * c_i * x^{i-1}$$

7. Sobrecargue (anulación u override) el método string __str__ en Python o el ToString en C#, para imprimir en la consola los polinomios en un formato más apropiado.
   a. Por ejemplo, para la lista o el arreglo de coeficientes de un polinomio dado por los elementos $[1, \quad 0, \quad 0, \quad 1, \quad 6]$, en lugar de imprimir el polinomio como:

      + 1*x^0 + 0*x^1 + 0*x^2 + –1*x^3 + –6*x^4

      Imprímalo como de la siguiente forma que es más apropiada:

      1 – x^3 – 6*x^4

Código Python – Visual Studio Code (Logo Azul): Declaración de la clase Polynomial

```
# -*- coding: utf-8 -*-


#Comentario de una sola linea con el simbolo #, en Python para nada se deben poner acentos sino el programa
#puede fallar o imprimir raro en consola, la siguiente línea de código es para que no tenga error, pero aún
```

```
#así al poner un ángulo saldrá raro en consola, la línea debe ponerse tal cual como aparece y justo al inicio.

#POO: La programación orientada a objetos es una forma de estructurar programas y aplicaciones de tal forma que
#los datos y las operaciones con estos se agrupan en clases para que se puedan acceder a estos por medio de
#objetos.
#3.-Polynomial: Clase que sirve para crear ecuaciones polinomiales, evaluar su resultado si se sustituye el valor
#de x por un entero, sumar, multiplicar, derivar polinomios e imprimir su resultado en consola.

#DECLARACIÓN DE LA CLASE POLYNOMIAL:
class Polynomial:
    #CONSTRUCTOR O INICIALIZADOR DE LA CLASE: En él se declaran los atributos que se reutilizarán en los demás
    #métodos y que además, deben a fuerza de tener un valor.
    def __init__(self, coefficients):
    #El parámetro del constructor de la clase son los coeficientes, que se ingresan en forma de vector.
        #self es distinto a this, por eso no debe tener el mismo nombre que el parámetro que está recibiendo el
        #constructor, se puede renombrar la variable que llega a la clase por medio del constructor.
        #self lo que hace es referenciar métodos o datos de la clase donde nos encontramos, es una referencia
        #a lo que sea que pertenezca a esta clase.
        #El parámetro coefficients deberá ser una lista (vector), sino arrojará un error el programa.
        self.coeff = coefficients
        #Un polinomio es descrito como: p(x) = a0 + a1*x + a2*x^2 + a3*x^3 + ... + an*x^n, donde los
        #coeficientes del polinomio son: a0, a1, a2, a3, ..., an.
        #Por ejemplo, el polinomio: p(x) = 1-x^2+2x^3, tiene como coeficientes el vector: [1, 0, -1, 2]



    #EVALUAR EL RESULTADO SI SE SUSTITUYE EL VALOR DE X POR UN VALOR ENTERO: A través del método call se evalúa
    #un polinomio sustituyendo su variable x por un valor numérico.
    #MÉTODO CALL: Este método es propio del lenguaje de programación Python y para utilizarze se declarar el
    #nombre del objeto y sus parámetros, a diferencia de los demás donde se declara el nombre del objeto más el
    #nombre del método:
    # - ObjetoClase(parámetro1, parámetro2, ..., parámetro_n)                    Método call
    # - ObjetoClase.NombreMétodo(parámetro1, parámetro2, ..., parámetro_n)       Método cualquiera
    #Solo puede existir un método call por clase y este trabaja con los atributos del constructor y puede
    #incluir otros.
    def __call__(self, x):
        #Variable temporal que evalúa cada una de las multiplicaciones de los coeficientes del polinomio con la
        #variable x (que ahora tiene un valor numérico).
        sigma = 0
        for i in range(len(self.coeff)):
            sigma = sigma + self.coeff[i]*x**i
        return sigma
```

```
#SOBRECARGANDO LOS OPERADORES: Esto se refiere a asignar un método a los símbolos de suma, resta,
#multiplicación, etc. para realizar en este caso operaciones matemáticas con los polinomios, esto se realiza
#a través de los siguientes nombres de métodos:
# - __add__: Corresponde al símbolo +
# - __sub__: Corresponde al símbolo -
# - __mul__: Corresponde al símbolo *
# - __str__: Corresponde a cuando se usa el método print() o str()

#MÉTODO __ADD__: Método para sumar polinomios.
def __add__(self, other):
    #Si el primer polinomio en la suma es mayor en tamaño que el segundo, entonces ese es el que se debe
    #asignar a la variable result_coeff, ya que en el bucle for que se incluye dentro, se recorrerán solo
    #los coeficientes del polinomio menor para realizar la operación, así dejando sin tocar a los polinomios
    #de mayor grado.
    if(len(self.coeff) > len(other.coeff)):             #Polinomio_1 > Polinomio_2
        #[:]: Recordemos que el valor de coeff es una lista, con la instrucción [indice_inicio:indice_final],
        #se puede crear una copia de estos elementos y asignarlos a otra variable, obteniendo así un
        #subconjunto de la lista, pero si a esta instrucción no se indica un índice de inicio y final, se
        #copian todos los elementos.
        result_coeff = self.coeff[:]                    #result_coeff = Polinomio_1
        #Bucle que recorre los elementos del polinomio de menor tamaño y los suma con los elementos del
        #polinomio de mayor grado.
        for i in range(len(other.coeff)):
            result_coeff[i] += other.coeff[i]           #result_coeff = Polinomio_1 + Polinomio_2
    #Si el primero polinomio en la suma es de menor tamaño que el segundo, entonces ese es el que se
    #asigna a la variable result_coeff, ya que en el bucle for que se incluye dentro, se recorrerán solo los
    #coeficientes del polinomio menor para realizar la operación, así dejando sin tocar a los polinomios de
    #mayor grado.
    else:                                               #Polinomio_2 > Polinomio_1
        result_coeff = other.coeff[:]                   #result_coeff = Polinomio_2
        #Bucle que recorre los elementos del polinomio de menor tamaño y los suma con los elementos del
        #polinomio de mayor grado.
        for i in range(len(self.coeff)):
            result_coeff[i] += self.coeff[i]            #result_coeff = Polinomio_2 + Polinomio_1
    return Polynomial(result_coeff)



#MÉTODO __SUB__: Método para restar polinomios.
def __sub__(self, other):
    #[:]: Recordemos que el valor de coeff es una lista, con la instrucción [indice_inicio:indice_final],
    #se puede crear una copia de estos elementos y asignarlos a otra variable, obteniendo así un subconjunto
    #de la lista, pero si a esta instrucción no se indica un índice de inicio y final, se copian todos los
    #elementos.
    result_coeff = self.coeff[:]                        #result_coeff = Polinomio_1
```

```
    #Condicional que ajusta la longitud de la lista de coeficientes del segundo polinomio agregando ceros al
    #final si es necesario cuando este es de menor tamaño, osea de menor grado, esto para que ambos
    #polinomios tengan la misma cantidad de términos para poder restar los coeficientes correctamente en el
    #siguiente bucle for.
    if len(other.coeff) < len(result_coeff):            #Polinomio_1 > Polinomio_2
        #Primero se calcula la diferencia de tamaños entre las listas de coeficientes:
        #   len(result_coeff) - len(other.coeff) = Tamaño Polinomio_1 - Tamaño Polinomio_2
        #Esta resta siempre será positiva, confirmando que el segundo polinomio tiene menos términos que el
        #primero.
        #Ahora multiplicamos una lista de ceros [0] por la diferencia de longitudes utilizando la expresión:
        #   [0] * (Tamaño Polinomio_1 - Tamaño Polinomio_2).
        #Esto crea una lista de ceros con una cantidad igual a la diferencia de longitudes.
        #Por ejemplo, si la diferencia es 2, se generará el vector [0, 0] y por medio de la operación += se
        #agregarán los ceros al final de la lista del Polinomio_2 para que tenga la misma longitud que
        #Polinomio_1.
        other.coeff += [0] * (len(result_coeff) - len(other.coeff))  #Tamaño Polinomio_2 = Tamaño Polinomio_1
    elif len(other.coeff) > len(result_coeff):          #Polinomio_2 > Polinomio_1
        #Se repite el mismo proceso hecho con el Polinomio_2 cuando Polinomio_1 > Polinomio_2, pero ahora se
        #aplica al Polinomio_1, ya que en este caso es de menor tamaño que el Polinomio_2.
        result_coeff += [0] * (len(other.coeff) - len(result_coeff)) #Tamaño Polinomio_1 = Tamaño Polinomio_2
    #Ya habiendo hecho igual el tamaño de los coeficientes de ambos polinomios, se realiza la resta:
    for i in range(len(result_coeff)):
        result_coeff[i] -= other.coeff[i]               #Polinomio_1 - Polinomio_2
    return Polynomial(result_coeff)



#MÉTODO __MUL__: Método para multiplicar polinomios.
def __mul__(self, other):
    c = self.coeff                      #Polinomio_1
    d = other.coeff                     #Polinomio_2
    #len(): Método que devuelve la longitud de una lista.
    M = len(c) - 1        #Número de índices del polinomio, por eso es que se resta un 1 al tamaño.
    N = len(d) - 1        #Número de índices del polinomio, por eso es que se resta un 1 al tamaño.
    import numpy as np #Librería numpy: Realiza operaciones matemáticas complejas (matriciales).
    #numpy.zeros(): Método que crea un vector o matriz lleno de ceros, en el primer parámetro se indica el
    #tamaño del eje x (número de columnas), en el segundo el tamaño del eje y (número de filas) y en el
    #terceroel tipo de dato de los elementos que conforman la matriz, los demás parámetros del método casi
    #no se usan.
    #Se crea un vector con la suma de ambos índices de los coeficientes de ambos polinomios + 1, para así
    #obtener un tamaño donde se pueda almacenar el resultado de la multiplicación de polinomios. Cada índice
    #corresponde al grado máximo (exponente máximo) de cada polinomio.
    result_coeff = np.zeros( M + N + 1) #Vector resultante de la multiplicación
```

```
        #Recorre los índices del primer polinomio partiendo desde cero
        for i in range(M + 1):              #Recorrer todos los coeficientes del Polinomio_1
            #Recorre los índices del segundo polinomio partiendo desde cero
            for j in range(N + 1):          #Recorrer todos los coeficientes del Polinomio_2
                #Dentro de este bucle for se calcula el producto de los coeficientes correspondientes a los
                #dos polinomios, es decir, c[i] * d[j].
                #Luego se suma este producto al coeficiente correspondiente al vector resultante en su índice i+j.
                #Esto se hace para acumular los términos de la multiplicación en la posición adecuada dentro del
                #polinomio resultante.
                result_coeff[i+j]=result_coeff[i+j]+c[i]*d[j]
        return Polynomial(result_coeff)



    #MÉTODO DERIVATE: Método para obtener la derivada de un polinomio, para ello se apoya en un método que obtiene
    #su diferencial.
    #differentiate(): Funcion para obtener el diferencial de un polinomio.
    def differentiate(self):
        #Recorre los índices del polinomio partiendo desde 1 hasta el número de su tanaño, que siempre es 1 más
        #al número de índices, se recorre desde 1 para que se pueda seguir la fórmula del diferencial que es la sig:
        #d/dx((i=0;i=n)∑ci*x) = (i=1;i=n)∑i*ci*x^(i-1)
        for i in range(1, len(self.coeff)): #Recorrer todos los coeficientes del Polinomio_1
            #d/dx = (i=1;i=n)∑i*ci*x^(i-1); Recordemos que el grado de x^(i-1) se accede a través del índice del
            #coeficiente perteneciente al polinomio.
            self.coeff[i-1]=i*self.coeff[i]
        #del: Permite eliminar uno o varios elementos de una lista, e incluso la misma lista. Esto se hace porque la
        #derivada reduce el grado máximo del polinomio en uno y se realiza utilizando el método del indicando la
        #posición [-1], el índice -1 en Python se utiliza para hacer referencia al último elemento de una lista.
        del self.coeff[-1]
    #derivate(): Funcion para derivar un polinomio
    def derivate(self):
        #Objeto o Instancia de la clase Polynomial: Recibe el parámetros del constructor de la clase Polynomial, que
        #es simplemente el vector de coeficientes del polinomio.
        dpdx = Polynomial(self.coeff[:])
        #Luego se le aplica el método differentiate() al objeto y de esta forma se obtiene la derivada del polinomio.
        dpdx.differentiate()
        return dpdx



    #MÉTODO __STR__: Función para imprimir el resultado de la operación de polinomios correctamente en consola.
    def __str__(self):
        s="" #String s vacío que se utilizará para construir la representación del polinomio en consola.
        #Recorre los índices del polinomio partiendo desde cero
        for i in range(len(self.coeff)):
```

```
            #Dentro del bucle for, se verifica si el coeficiente en la posición actual es diferente de cero, esto
            #porque los coeficientes que tengan un cero no aparecerán en la representación del polinomio.
            if(self.coeff[i] != 0):
                #Si el coeficiente no es cero, se agrega un string a la variable "s" que represente el término
                #correspondiente a la multiplicación de la constante (coeficiente) por la variable del polonomio
                #elevada al exponente que le corresponde, ya que los polinomios son descritos como:
                #p(x) = a0 + a1*x + a2*x^2 + a3*x^3 + ... + an*x^n,
                #donde los coeficientes del polinomio son: a0, a1, a2, a3, ..., an.
                #Por ejemplo, el polinomio: p(x) = 1-x^2+2x^3, tiene como coeficientes el vector: [1, 0, -1, 2]
                s += " + %g*x^%d" %(self.coeff[i],i)
            #replace(): Método que reemplaza un caracter que se encuentra en un string por otro declarado por
            #nosotros, esto se ejecutará todas las veces que dicho caracter aparezca en el string.
            #A continuación, se realizan una serie de reemplazos en la cadena s utilizando el método replace() para
            #mejorar la presentación del polinomio:
            s= s.replace("+ -","- ")    #"+ -" se reemplaza por "- " para eliminar la redundancia de signos.
            s= s.replace("x^0","1")    #"x^0" se reemplaza por " " para representar el término constante.
            s= s.replace("1*", " ")     #"1*" se reemplaza por " " para simplificar la representación de una constante.
            s= s.replace("x^1","x")     #"x^1" se reemplaza por "x" para representar x elevada a la primera potencia.
            #Después de realizar los reemplazos, se verifican algunas condiciones en la cadena s para ajustar el
            #formato:
            if(s[0:3]==" + "):
                #Si los primeros tres caracteres de s son " + ", estos se eliminan para evitar que aparezca un signo +
                #innecesario al comienzo de la expresión.
                s = s[3:]
            if(s[0:3]==" - "):
                #Si los primeros tres caracteres de s son " - ", se reemplazan por un "-" para tener un signo menos al
                #comienzo de la expresión cuando esta sea negativa.
                s = "-" + s[3:]
        return s
```

Código Python – Visual Studio Code (Logo Azul): Uso de la clase Polynomial

```
# -*- coding: utf-8 -*-

#Comentario de una sola linea con el simbolo #, en Python para nada se deben poner acentos sino el programa
#puede fallar o imprimir raro en consola, la siguiente línea de código es para que no tenga error, pero aún
#así al poner un ángulo saldrá raro en consola, la línea debe ponerse tal cual como aparece y justo al inicio.

#Importa de la clase Acount del archivo que originalmente era 11.-POO_Cuenta y se cambió porque Python no
#puede manejar bien este nombre de archivo para importar
from Archivos_Ejercicios_Python.POO_17_3_Polynomial.POO_Polinomio_Declaración import Polynomial

#Objetos o Instancias de la clase Polynomial: Recibe el parámetros del constructor de la clase Polynomial,
#que es el vector (lista) de coeficientes del polinomio.
p1 = Polynomial([1,-1])
```

```
p2 = Polynomial([0,1,0,0,-6,-1])
#print(): Imprimir un mensaje en consola, la forma de concatenar este mensaje puede ser añadiendo un símbolo
#de +  una coma entre el string que se quiere concatenar y las variables.
print("p1(x) =", p1)
print("p2(x) =", p2)

#SOBRECARGANDO LOS OPERADORES: Esto se refiere a asignar un método a los símbolos de suma, resta, multiplicación,
#etc. para realizar en este caso operaciones matemáticas con los polinomios.

#MÉTODO __ADD__: Método para sumar polinomios, corresponde al símbolo +.
p3 = p1 + p2
print("p3 = p1 + p2 =", p3)

#MÉTODO __SUB__: Método para restar polinomios, corresponde al símbolo -.
p4 = p1 - p2
print("p4 = p1 - p2 =", p4)

#MÉTODO __MUL__: Método para multiplicar polinomios, corresponde al símbolo *.
p5 = p1 * p2
print("p5 = p1 * p2 =", p5)

#MÉTODO DERIVATE: Método para obtener la derivada de un polinomio.
p6 = p2.derivate()
print("dp2/dx =", p6)

#EVALUAR EL RESULTADO DE UN POLINOMIO SI SE SUSTITUYE LA VARIABLE X POR UN VALOR ENTERO: A través del método
#call se evalúa un polinomio sustituyendo su variable x por un valor numérico, para ello se debe declarar el
#valor de x, usar una instancia de la clase y posteriormente pasar como parámetro la variable x.
x = 1
#print(): Imprimir un mensaje en consola, la forma de concatenar este mensaje puede ser añadiendo un símbolo
#de +  una coma entre el string que se quiere concatenar y las variables o además se puede utilizar la
#nomenclatura de %g, donde se coloca este signo cada que se quiera colocar el valor de una variable entre un
#string (denotado entre comillas), luego se coloca un signo de % y finalmente un paréntesis con las variables
#u objetos que se quiera concatenar, separados entre comillas y colocados en el órden en el que se hayan
#colocado en el string.
print("p1(x=%g) = %g"%(x, p1(x)))
print("p2(x=%g) = %g"%(x, p2(x)))
```

Resultado del Código Python

```
PS C:\Users\diego\OneDrive\Documents\Aprendiendo\Python\1.-Instrumentación Virtual> & D:/Users/diego/AppData/Local/Programs/Python/Python39/python.exe
"c:/Users/diego/OneDrive/Documents/Aprendiendo/Python/1.-Instrumentación Virtual/17.3.-POO Polinomio Uso.py"
p1(x) =  1 -  x
p2(x) =  x - 6*x^4 -  x^5
p3 = p1 + p2 =  1 - 6*x^4 -  x^5
p4 = p1 - p2 =  1 - 2*x + 6*x^4 +  x^5
p5 = p1 * p2 =  x -  x^2 - 6*x^4 + 5*x^5 +  x^6
dp2/dx =  1 - 24*x^3 - 5*x^4
p1(x=1) = 0
p2(x=1) = -6
PS C:\Users\diego\OneDrive\Documents\Aprendiendo\Python\1.-Instrumentación Virtual>
```

Código C# (.NET Framework) – Visual Studio (Logo Morado)

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Class_Polynomial
{
     class Polynomial
     {
          public readonly double[] coeff;
          string s;
          public Polynomial(double[] coeff)
          {
               this.coeff = coeff;
          }// Costructor
          public double Eval(double x)
          {
               double result = 0;
               for (int i = 0; i < coeff.Length; i++)
               {
                    result = result + coeff[i] * Math.Pow(x, i);
               }
               return result;
          }//Evaluar el polinomio
          public static Polynomial operator +(Polynomial a, Polynomial b)
          {
               double[] result;
               if (a.coeff.Length > b.coeff.Length)
               {
                    result = new double[a.coeff.Length];
                    for (int i = 0; i < a.coeff.Length; i++)
                    {
                         result[i] = a.coeff[i];
                    }
                    for (int i = 0; i < b.coeff.Length; i++)
                    {
                         result[i] += b.coeff[i];
                    }
               }
               else
               {
                    result = new double[b.coeff.Length];
                    for (int i = 0; i < b.coeff.Length; i++)
                    {
                         result[i] = b.coeff[i];
                    }
                    for (int i = 0; i < a.coeff.Length; i++)
                    {
                         result[i] += a.coeff[i];
                    }
               }
               return new Polynomial(result);
          }//Sobrecarga del operador +
          public static Polynomial operator *(Polynomial a, Polynomial b)
          {
               double[] c = new double[a.coeff.Length];
               for (int i = 0; i < a.coeff.Length; i++)
               {
                    c[i] = a.coeff[i];
               }
               double[] d = new double[b.coeff.Length];
               for (int i = 0; i < b.coeff.Length; i++)
               {
                    d[i] = b.coeff[i];
               }
               int M = c.Length - 1;
               int N = d.Length - 1;
               double[] resultCoeff = new double[M + N + 1];
               for (int i = 0; i < resultCoeff.Length; i++)
               {
                    resultCoeff[i] = 0;
               }
               for (int i = 0; i < M + 1; i++)
               {
                    for (int j = 0; j < N + 1; j++)
                    {
```

```
                    resultCoeff[i + j] += c[i] * d[j];
                }
            }
            return new Polynomial(resultCoeff);
        }//Sobrecarga del operador *
        private double[] Differentiate(Polynomial a)
        {
            double[] result = new double[a.coeff.Length];
            for (int i = 0; i < a.coeff.Length; i++)
            {
                result[i] = a.coeff[i];
            }
            for (int i = 1; i < result.Length; i++)
            {
                result[i - 1] = i * result[i];
            }
            Array.Resize(ref result, result.Length - 1);
            return result;
        }
        public Polynomial Derivative()
        {
            double[] cd = new double[this.coeff.Length];
            for (int i = 0; i < this.coeff.Length; i++)
            {
                cd[i] = this.coeff[i];
            }
            Polynomial e = new Polynomial(cd);
            double[] result;
            result = Differentiate(e);
            return new Polynomial(result);
        }
        public override string ToString()
        {
            s = "";
            for (int i = 0; i < this.coeff.Length; i++)
            {
                if (this.coeff[i] != 0.0)
                {
                    s += " + " + Convert.ToString(this.coeff[i])
                        + "*x^" + Convert.ToString(i);
                }
            }
            s = s.Replace("+ -", "- ");
            s = s.Replace("x^0", "1");
            s = s.Replace("1*", " ");
            s = s.Replace("x^1", "x");
            if (s.Substring(0, 3) == " + ")
            {
                s = s.Substring(3);
            }
            if (s.Substring(0, 3) == " - ")
            {
                s = "-" + s.Substring(3);
            }
            return s;
        }
    }
    class Program
    {
        static string ShowCoeff(Polynomial p)
        {
            string r = "[";
            for (int i = 0; i < p.coeff.Length; i++)
            {
                r += Convert.ToString(p.coeff[i]) + ",";
            }
            r += Convert.ToString(p.coeff[p.coeff.Length - 1]) + "]";
            return r;
        }
        static void Main(string[] args)
        {
            double[] cp1 = new double[] { 1, -1 };
            Polynomial p1 = new Polynomial(cp1);
            double[] cp2 = new double[] { 0, 1, 0, 0, -6, -1 };
            Polynomial p2 = new Polynomial(cp2);
            double x = 1;
            Console.WriteLine("P1(x={0} = {1}", x, p1.Eval(x));
            Console.WriteLine("P2(x={0} = {1}", x, p2.Eval(x));
            Polynomial p3;
```

```
                p3 = p1 + p2;
                Console.WriteLine("p3 = p1+p2 = {0}", ShowCoeff(p3));
                Polynomial p4;
                p4 = p1 * p2;
                Console.WriteLine("p4 = p1*p2 = {0}", ShowCoeff(p4));
                Polynomial p5;
                p5 = p2.Derivative();
                Console.WriteLine("p5 = dp2/dx = {0}", ShowCoeff(p5));
                Console.WriteLine("p1(x)={0}", p1);
                Console.WriteLine("p2(x)={0}", p2);
                Console.WriteLine("p3(x)={0}", p3);
                Console.WriteLine("p4(x)={0}", p4);
                Console.WriteLine("p5(x)={0}", p5);
            }
        }
}
```

Resultado del Código C#