Sistemas de Gestión Empresarial

Ejercicios Clases Python

Mattias Nygren Jiménez
1-10-2025
2º DAM

Índice

1. Ejercicio 1. Fracciones

Código:

```python
from math import gcd

class Fraccion:

    def __init__(self, num, den):
        self.numerador = num
        self.denominador = den
        if self.denominador == 0:
            raise ZeroDivisionError("Denominator cannot be zero.")

        self.signo = -1 if self.numerador * self.denominador < 0 else 1

        self.numerador, self.denominador = abs(self.numerador), abs(self.denominador)

        divisor = gcd(self.numerador, self.denominador)
        self.numerador //= divisor
        self.denominador //= divisor

    def __repr__(self):
        signo_str = "-" if self.signo < 0 else ""
        return signo_str + str(self.numerador) + '/' + str(self.denominador)

    def __eq__(self, fb):
        return (self.signo == fb.signo and
                self.numerador == fb.numerador and
                self.denominador == fb.denominador)

    def __add__(self, fb):
        num1 = self.signo * self.numerador
        num2 = fb.signo * fb.numerador
        nuevo_num = num1 * fb.denominador + num2 * self.denominador
        nuevo_den = self.denominador * fb.denominador
        return Fraccion(nuevo_num, nuevo_den)

    def __lt__(self, fb):
        return (self.signo * self.numerador * fb.denominador <
                fb.signo * fb.numerador * self.denominador)

    def __le__(self, fb):
        return self < fb or self == fb

    def __gt__(self, fb):
        return not (self <= fb)

    def __ge__(self, fb):
        return not (self < fb)

```

```
47
48  class FraccionEnt(Fraccion):
49
50      def __init__(self,num,den,ent):
51          super().__init__(num,den)
52          pentera=ent
53
54  x=Fraccion(1,2)
55  print(x.numerador)
56
57
58  f1 = Fraccion(2, 4)
59  f2 = Fraccion(1, 2)
60  f3 = Fraccion(3, 4)
61
62  print(f1)
63  print(f1 == f2)
64  print(f1 + f3)
65  print(f1 < f3)
66  print(f1 <= f2)
67  print(f3 > f1)
68  print(f2 >= f1)
```

Ejecución:

```
1
1/2
True
5/4
True
True
True
```

2. Ejercicio 2. Figuras

Código:

```python
import math

class FiguraGeometrica:
    def superficie(self):
        return 0

class TrianguloRectangulo(FiguraGeometrica):
    def __init__(self, cateto1, cateto2):
        self.cateto1 = cateto1
        self.cateto2 = cateto2

    def hipotenusa(self):
        return math.sqrt(self.cateto1**2 + self.cateto2**2)

    def superficie(self):
        return (self.cateto1 * self.cateto2) / 2


class Rectangulo(FiguraGeometrica):
    def __init__(self, base, altura):
        self.base = base
        self.altura = altura

    def superficie(self):
        return self.base * self.altura

class ListaDeFiguras:
    def __init__(self):
        self.figuras = []

    def añadir_triangulo(self, cateto1, cateto2):
        self.figuras.append(TrianguloRectangulo(cateto1, cateto2))

    def añadir_cuadrado(self, lado):
        self.figuras.append(Rectangulo(lado, lado))

    def superficie_total(self):
        return sum(figura.superficie() for figura in self.figuras)

    def contar_triangulos(self):
        return sum(isinstance(figura, TrianguloRectangulo) for figura in self.figuras)

```

```
43
44  lista = ListaDeFiguras()
45
46  t = TrianguloRectangulo(3, 4)
47  r = Rectangulo(2, 5)
48
49  print(t.hipotenusa())
50  print(t.superficie())
51  print(r.superficie())
52
53  lista.añadir_triangulo(3, 4)
54  lista.añadir_cuadrado(4)
55
56  print(lista.superficie_total())
57  print(lista.contar_triangulos())
```

Ejecución:

```
5.0
6.0
10
22.0
1
```

3. Ejercicio 3. Donuts

Código:

```python
from random import randrange

class Donuts:
    def __init__(self):
        self.tablero = []
        self.jugadaAnterior = None
        self.victoria = 0

    def generarTablero(self):
        # Genera un array con números del 0 al 3, cada uno corresponde con una dirección
        self.tablero = [[randrange(0, 4) for _ in range(6)] for _ in range(6)]

    def pintarTablero(self):
        # Pinta el tablero con diseño ASCII, según el array generado
        n = 1
        print("\n\t    1 2 3 4 5 6")
        print("\t  ╔═══════════╗")
        for row in self.tablero:
            linea = f"\t{n} ‖ "
            for cell in row:
                match cell:
                    case 0:
                        linea += "| "
                    case 1:
                        linea += "– "
                    case 2:
                        linea += "/ "
                    case 3:
                        linea += "\\ "
                    case 4 | 5 | 6 | 7:
                        linea += "x "
                    case 8 | 9 | 10 | 11:
                        linea += "o "
            n+=1
            linea += "‖ "
            print(linea)
        print("\t  ╚═══════════╝")

        if self.jugadaAnterior != None:
            match self.tablero[self.jugadaAnterior[0]][self.jugadaAnterior[1]]:
                case 0 | 4 | 8:
                    casilla = "|"
                case 1 | 5 | 9:
                    casilla = "–"
                case 2 | 6 | 10:
                    casilla = "/"
                case 3 | 7 | 11:
                    casilla = "\\"

            print(f"\n\tÚltima casilla: {casilla} ({self.jugadaAnterior[0]+1}, {self.jugadaAnterior[1]+1})")
```

```python
    def verificarContiguas(self, y, x):
        # Devuelve True si (y,x) es contigua a la última ficha según la dirección de la casilla
        if self.jugadaAnterior is None:
            return True

        y0, x0 = self.jugadaAnterior
        valor_anterior = self.tablero[y0][x0] % 4
        direcciones = {
            0: [(-1,0),(1,0)],       # vertical
            1: [(0,-1),(0,1)],       # horizontal
            2: [(-1,1),(1,-1)],      # diagonal /
            3: [(-1,-1),(1,1)]       # diagonal \
        }

        for dy, dx in direcciones[valor_anterior]:
            ny, nx = y0 + dy, x0 + dx
            if 0 <= ny < 6 and 0 <= nx < 6 and self.tablero[ny][nx] in (0,1,2,3):
                if (ny, nx) == (y, x):
                    return True
        return False
```

```python
    def verificarLineaBorde(self, y, x):
        # Devuelve True si (y,x) está en la línea libre de un borde según la dirección de la última ficha
sers\matti\Desktop\git\sge\clases-python\3.py
        erior is None:
            return False

        y0, x0 = self.jugadaAnterior
        valor_anterior = self.tablero[y0][x0] % 4
        direcciones = {
            0: [(-1,0),(1,0)],
            1: [(0,-1),(0,1)],
            2: [(-1,1),(1,-1)],
            3: [(-1,-1),(1,1)]
        }

        dir1, dir2 = direcciones[valor_anterior]

        def recorrer(dy, dx):
            ny, nx = y0 + dy, x0 + dx
            while 0 <= ny < 6 and 0 <= nx < 6:
                if self.tablero[ny][nx] in (0,1,2,3):
                    if (ny, nx) == (y, x):
                        return True
                ny += dy
                nx += dx
            return False

        dy1, dx1 = dir1
        dy2, dx2 = dir2
        bloqueada1 = not (0 <= y0 + dy1 < 6 and 0 <= x0 + dx1 < 6)
        bloqueada2 = not (0 <= y0 + dy2 < 6 and 0 <= x0 + dx2 < 6)

        if bloqueada1 and not bloqueada2:
            return recorrer(dy2, dx2)
        if bloqueada2 and not bloqueada1:
            return recorrer(dy1, dx1)

        return False
```

```python
    def verificarCasilla(self, y, x):
        """
        Devuelve True si la jugada (y,x) es válida según el orden de prioridades:
        - Casillas contiguas
        - Línea de casillas desde borde
        - Libertad total
        """
        if self.tablero[y][x] not in (0,1,2,3):
            return False
        if self.verificarContiguas(y,x):
            return True
        if self.verificarLineaBorde(y,x):
            return True
        return False

    def movimientosValidos(self):
        posiciones = []

        # Contiguas
        for y in range(6):
            for x in range(6):
                if self.verificarContiguas(y,x):
                    posiciones.append((y,x))
        if posiciones:
            return posiciones

        # Línea desde borde
        for y in range(6):
            for x in range(6):
                if self.verificarLineaBorde(y,x):
                    posiciones.append((y,x))
        if posiciones:
            return posiciones

        # Libertad total
        for y in range(6):
            for x in range(6):
                if self.tablero[y][x] in (0,1,2,3):
                    posiciones.append((y,x))

        return posiciones

```

```python
    def colocar(self, n, y, x):
        # Comprobación contigua
        if not (0 <= x < 6 and 0 <= y < 6):
            return False

        # Comprobación casilla vaía
        if self.tablero[y][x] not in (0,1,2,3):
            return False

        # Si es la primera jugada, cualquier casilla es válida
        if self.jugadaAnterior is None:
            permitir = True
        else:
            validas = self.movimientosValidos()
            permitir = (y, x) in validas

        if not permitir:
            return False

        match self.tablero[y][x]:
            case 0 | 4 | 8:
                self.tablero[y][x] = 4
            case 1 | 5 | 9:
                self.tablero[y][x] = 5
            case 2 | 6 | 10:
                self.tablero[y][x] = 6
            case 3 | 7 | 11:
                self.tablero[y][x] = 7
        if n == 2:
            self.tablero[y][x] += 4

        return True

```

```python
    def verificarVictoria(self):
        self.victoria = 0

        def det_negro(val):
            return val in (4,5,6,7)
        def det_blanco(val):
            return val in (8,9,10,11)

        # vertical, horizontal, diagonal \, diagonal /
        direcciones = [(1,0), (0,1), (1,1), (1,-1)]

        for y in range(6):
            for x in range(6):
                val = self.tablero[y][x]
                if not (det_negro(val) or det_blanco(val)):
                    continue

                # determinar a qué jugador pertenece esta celda
                es_j1 = det_negro(val)

                for dy, dx in direcciones:
                    contador = 1
                    ny, nx = y + dy, x + dx
                    while 0 <= ny < 6 and 0 <= nx < 6:
                        nv = self.tablero[ny][nx]
                        if es_j1 and det_negro(nv):
                            contador += 1
                        elif (not es_j1) and det_blanco(nv):
                            contador += 1
                        else:
                            break
                        ny += dy
                        nx += dx

                    if contador >= 5:
                        self.victoria = 1 if es_j1 else 2
                        return
```

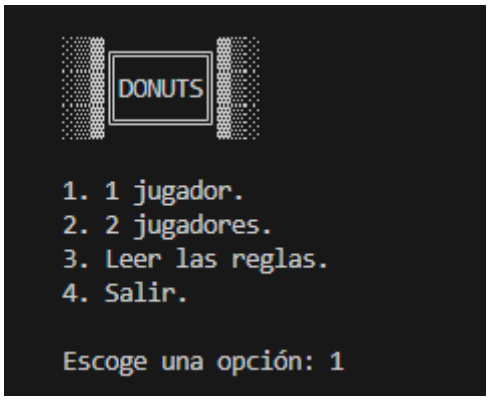
```python
    def tableroLleno(self):
        # Detecta si al tablero no le queda ninguna casilla vacía
        return all(celda not in (0, 1, 2, 3) for fila in self.tablero for celda in fila)

    def capturarFichas(self, jugador, y, x):
        direcciones = [
            (-1, 0), (1, 0),       # vertical
            (0, -1), (0, 1),       # horizontal
            (-1, -1), (-1, 1),     # diagonales
            (1, -1), (1, 1)
        ]

        if jugador == 1:
            propio = (4, 5, 6, 7)
            rival = (8, 9, 10, 11)
        else:
            propio = (8, 9, 10, 11)
            rival = (4, 5, 6, 7)

        for dy, dx in direcciones:
            ny, nx = y + dy, x + dx
            fichas_capturadas = []

            while 0 <= ny < 6 and 0 <= nx < 6 and self.tablero[ny][nx] in rival:
                fichas_capturadas.append((ny, nx))
                ny += dy
                nx += dx

            if 0 <= ny < 6 and 0 <= nx < 6 and self.tablero[ny][nx] in propio and len(fichas_capturadas) > 0:
                for cy, cx in fichas_capturadas:
                    forma = self.tablero[cy][cx] % 4
                    if jugador == 1:
                        self.tablero[cy][cx] = 4 + forma
                    else:
                        self.tablero[cy][cx] = 8 + forma
```

```
260     def empezarJuego(self, nJ):
261         self.jugadaAnterior = None
262         self.generarTablero()
263         self.pintarTablero()
264         while self.victoria == 0:
265             valido = False
266             while valido == False:
267                 print("\n\t▀▄ Donuts negros ▄▀")
268                 try:
269                     y = int(input("\tCoordenada vertical: ")) - 1
270                     x = int(input("\tCoordenada horizontal: ")) - 1
271                 except:
272                     y = -1
273                     x = -1
274                 if self.jugadaAnterior != None:
275                     valido = self.colocar(1, y, x)
276                 else:
277                     self.colocar(1, y, x)
278                     valido = True
279                 if valido == True:
280                     self.jugadaAnterior = (y, x)
281                     self.capturarFichas(1, y, x)
282                     self.pintarTablero()
283                 else:
284                     print("\n\tCasilla no válida.")
285             valido = False
286             if nJ == 1:
287                 validas = self.movimientosValidos()
288                 if validas:
289                     y, x = validas[randrange(len(validas))]
290                     self.colocar(2, y, x)
291                     self.jugadaAnterior = (y, x)
292                     self.capturarFichas(2, y, x)
293                     print("\n\t▒▒ Donuts blancos (IA) ▒▒")
294                     self.pintarTablero()
295                 else:
296                     print("\n\t¡Empate!")
297                     break
```

```
298                 else:
299                     while valido == False:
300                         print("\n\t Donuts blancos ")
301                         try:
302                             y = int(input("\tCoordenada vertical: ")) - 1
303                             x = int(input("\tCoordenada horizontal: ")) - 1
304                         except:
305                             y = -1
306                             x = -1
307                         valido = self.colocar(2, y, x)
308                         if valido == True:
309                             self.jugadaAnterior = (y, x)
310                             self.capturarFichas(2, y, x)
311                             self.pintarTablero()
312                         else:
313                             print("\n\tCasilla no válida.")
314                 self.verificarVictoria()
315 
316                 if self.tableroLleno():
317                     print("\n\t¡Empate!")
318                     break
319             if self.victoria == 1:
320                 print("\n¡Ganan los donuts negros!")
321             if self.victoria == 2:
322                 print("\n¡Ganan los donuts blancos!")
```

```
326 class Menu:
327     def __init__(self):
328         self.sel = 0
329         self.juego = Donuts()
330     def iniciar(self):
331         while not (self.sel == 4):
332             print("\n\t ┌──────┐\n\t │DONUTS│\n\t └──────┘")
333             print("\n\t1. 1 jugador.")
334             print("\t2. 2 jugadores.")
335             print("\t3. Leer las reglas.")
336             print("\t4. Salir.")
337             try:
338                 self.sel = int(input("\n\tEscoge una opción: "))
339             except:
340                 self.sel = 0
341             match self.sel:
342                 case 1:
343                     self.juego.empezarJuego(1)
344                 case 2:
345                     self.juego.empezarJuego(2)
346                 case 3:
347                     print("\n\tDonuts es un juego similar al 3 en raya.")
348                     print("\tDebes colocar 5 donuts consecutivos en una línea antes que tu oponente.")
349                     print("\n\tSe genera un tablero nuevo aleatorio cada partida.")
350                     print("\tSolo puedes colocar tu donut en las casillas contiguas la última casilla ocupada por tu oponente, en la dirección marcada por la línea.")
351                     print("\t| = arriba o abajo  - = izquierda o derecha  / = arriba a la derecha o abajo a la izquierda  \\ = arriba a la izquierda o abajo a la derecha")
352                     print("\n\tSi colocas dos donuts a los lados de tu contrincante, su ficha pasa a ser tuya.")
353                     print("\tSi un jugador coloca un donut en un borde del tablero, el contrincante puede colocar su donut en toda la línea de casillas que indica la dirección, sin ser contigua.")
354                     print("\n\tPuedes jugar contra la IA o contra un amigo.")
355                 case 4:
356                     print("\n\tSaliendo del programa...")
357                 case _:
358                     print("\n\tERROR: opción no válida.")
359 
360 Menu().iniciar()
```

Ejecución:


DONUTS
1. 1 jugador.
2. 2 jugadores.
3. Leer las reglas.
4. Salir.
Escoge una opción: 1


1 2 3 4 5 6
1 / | - | \ /
2 / - / | \
3 \ - | | \ /
4 - \ / - | \
5 / \ / \ | /
6 - - \ | | \
Donuts negros
Coordenada vertical: 3
Coordenada horizontal: 3
1 2 3 4 5 6
1 / | - | \ /
2 / - / | \
3 \ - x | \ /
4 - \ / - | \
5 / \ / \ | /
6 - - \ | | \
Última casilla: | (3, 3)
Donuts blancos (IA)
1 2 3 4 5 6
1 / | - | \ /
2 / - o | \
3 \ - x | \ /
4 - \ / - | \
5 / \ / \ | /
6 - - \ | | \
Última casilla: / (2, 3)
Donuts negros
Coordenada vertical: