

Sistemas de Gestión Empresarial

Ejercicios Clases Python

Mattias Nygren Jiménez

2-12-2025

2º DAM

Índice

1.	Ejercicio 1. Fracciones.....	1
2.	Ejercicio 2. Figuras	3
3.	Ejercicio 3. Donuts	5

1. Ejercicio 1. Fracciones

Código:

```
1  from math import gcd
2
3  class Fraccion:
4
5      def __init__(self, num, den):
6          self.numerador = num
7          self.denominador = den
8          if self.denominador == 0:
9              raise ZeroDivisionError("Denominator cannot be zero.")
10
11         self.signo = -1 if self.numerador * self.denominador < 0 else 1
12
13         self.numerador, self.denominador = abs(self.numerador), abs(self.denominador)
14
15         divisor = gcd(self.numerador, self.denominador)
16         self.numerador //= divisor
17         self.denominador //= divisor
18
19     def __repr__(self):
20         signo_str = "-" if self.signo < 0 else ""
21         return signo_str + str(self.numerador) + '/' + str(self.denominador)
22
23     def __eq__(self, fb):
24         return (self.signo == fb.signo and
25                 self.numerador == fb.numerador and
26                 self.denominador == fb.denominador)
27
28     def __add__(self, fb):
29         num1 = self.signo * self.numerador
30         num2 = fb.signo * fb.numerador
31         nuevo_num = num1 * fb.denominador + num2 * self.denominador
32         nuevo_den = self.denominador * fb.denominador
33         return Fraccion(nuevo_num, nuevo_den)
34
35     def __lt__(self, fb):
36         return (self.signo * self.numerador * fb.denominador <
37                 fb.signo * fb.numerador * self.denominador)
38
39     def __le__(self, fb):
40         return self < fb or self == fb
41
42     def __gt__(self, fb):
43         return not (self <= fb)
44
45     def __ge__(self, fb):
46         return not (self < fb)
47
```

```
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
True
```

2. Ejercicio 2. Figuras

Código:

```
1  import math
2
3  class FiguraGeometrica:
4      def superficie(self):
5          return 0
6
7  class TrianguloRectangulo(FiguraGeometrica):
8      def __init__(self, cateto1, cateto2):
9          self.cateto1 = cateto1
10         self.cateto2 = cateto2
11
12         def hipotenusa(self):
13             return math.sqrt(self.cateto1**2 + self.cateto2**2)
14
15         def superficie(self):
16             return (self.cateto1 * self.cateto2) / 2
17
18
19  class Rectangulo(FiguraGeometrica):
20      def __init__(self, base, altura):
21          self.base = base
22          self.altura = altura
23
24         def superficie(self):
25             return self.base * self.altura
26
27  class ListaDeFiguras:
28      def __init__(self):
29          self.figuras = []
30
31         def añadir_triangulo(self, cateto1, cateto2):
32             self.figuras.append(TrianguloRectangulo(cateto1, cateto2))
33
34         def añadir_cuadrado(self, lado):
35             self.figuras.append(Rectangulo(lado, lado))
36
37         def superficie_total(self):
38             return sum(figura.superficie() for figura in self.figuras)
39
40         def contar_triangulos(self):
41             return sum(isinstance(figura, TrianguloRectangulo) for figura in self.figuras)
42
```

```
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:

```

1  from random import randrange
2
3  class Donuts:
4      def __init__(self):
5          self.tablero = []
6          self.jugadaAnterior = None
7          self.victoria = 0
8
9      def generarTablero(self):
10         # Genera un array con números del 0 al 3, cada uno corresponde con una dirección
11         self.tablero = [[randrange(0, 4) for _ in range(6)] for _ in range(6)]
12
13     def pintarTablero(self):
14         # Pinta el tablero con diseño ASCII, según el array generado
15         n = 1
16         print("\n\t\t\t 1 2 3 4 5 6")
17         print("\t\t\t ┌──────────┐")
18         for row in self.tablero:
19             linea = f"\t\t{n}\t │ │ "
20             for cell in row:
21                 match cell:
22                     case 0:
23                         linea += " │ "
24                     case 1:
25                         linea += "─ "
26                     case 2:
27                         linea += " / "
28                     case 3:
29                         linea += " \ "
30                     case 4 | 5 | 6 | 7:
31                         linea += "  "
32                     case 8 | 9 | 10 | 11:
33                         linea += "o "
34             n+=1
35             linea += " │ │ "
36             print(linea)
37         print("\t\t\t └──────────┘")
38

```

```

39         if self.jugadaAnterior != None:
40             match self.tablero[self.jugadaAnterior[0]][self.jugadaAnterior[1]]:
41                 case 0 | 4 | 8:
42                     casilla = "|"
43                 case 1 | 5 | 9:
44                     casilla = "-"
45                 case 2 | 6 | 10:
46                     casilla = "/"
47                 case 3 | 7 | 11:
48                     casilla = "\\"
49
50             print(f"\n\t\tÚltima casilla: {casilla} ({self.jugadaAnterior[0]+1}, {self.jugadaAnterior[1]+1})")
51
52     def movimientosValidos(self):
53         """
54         Devuelve una lista de posiciones válidas para colocar un donut, siguiendo el orden de prioridades:
55         - Línea de casillas completa
56         - Libertad total (si la línea está llena)
57         """
58         if self.jugadaAnterior is None:
59             # Primera jugada: cualquier casilla libre
60             return [(y, x) for y in range(6) for x in range(6) if self.tablero[y][x] in (0,1,2,3)]
61
62         y0, x0 = self.jugadaAnterior
63         valor_anterior = self.tablero[y0][x0] % 4
64         direcciones = {
65             0: [(-1,0),(1,0)],      # vertical
66             1: [(0,-1),(0,1)],      # horizontal
67             2: [(-1,1),(1,-1)],     # diagonal /
68             3: [(-1,-1),(1,1)]     # diagonal \
69         }
70
71         linea = []
72         for dy, dx in direcciones[valor_anterior]:
73             ny, nx = y0 + dy, x0 + dx
74             while 0 <= ny < 6 and 0 <= nx < 6:
75                 if self.tablero[ny][nx] in (0,1,2,3):
76                     linea.append((ny, nx))
77                 ny += dy
78                 nx += dx
79
80         if linea:
81             return linea
82
83         # Si la línea está llena -> Libertad total
84         return [(y, x) for y in range(6) for x in range(6) if self.tablero[y][x] in (0,1,2,3)]
85

```

```

86     def colocar(self, n, y, x):
87         if (y, x) not in self.movimientosValidos():
88             return False
89
90         match self.tablero[y][x]:
91             case 0 | 4 | 8:
92                 self.tablero[y][x] = 4
93             case 1 | 5 | 9:
94                 self.tablero[y][x] = 5
95             case 2 | 6 | 10:
96                 self.tablero[y][x] = 6
97             case 3 | 7 | 11:
98                 self.tablero[y][x] = 7
99         if n == 2:
100             self.tablero[y][x] += 4
101
102         return True
103
104     def verificarVictoria(self):
105         self.victoria = 0
106         # vertical, horizontal, diagonal \, diagonal /
107         direcciones = [(1,0), (0,1), (1,1), (1,-1)]
108         for y in range(6):
109             for x in range(6):
110                 val = self.tablero[y][x]
111                 if val not in (4,5,6,7,8,9,10,11):
112                     continue
113                 jugador = 1 if val in (4,5,6,7) else 2
114                 grupo = (4,5,6,7) if jugador == 1 else (8,9,10,11)
115                 for dy, dx in direcciones:
116                     contador = 1
117                     ny, nx = y + dy, x + dx
118                     while 0 <= ny < 6 and 0 <= nx < 6 and self.tablero[ny][nx] in grupo:
119                         contador += 1
120                         ny += dy
121                         nx += dx
122                     if contador >= 5:
123                         self.victoria = jugador
124                         return
125
126     def tableroLleno(self):
127         # Detecta si al tablero no le queda ninguna casilla vacía
128         return all(celda not in (0, 1, 2, 3) for fila in self.tablero for celda in fila)
129

```



```
130     def capturarFichas(self, jugador, y, x):
131         pares_direcciones = [
132             [(-1, 0), (1, 0)],      # vertical
133             [(0, -1), (0, 1)],      # horizontal
134             [(-1, -1), (1, 1)],     # diagonal \
135             [(1, -1), (-1, 1)]      # diagonal /
136         ]
137
138         if jugador == 1:
139             rival = (8, 9, 10, 11)
140         else:
141             rival = (4, 5, 6, 7)
142
143         for (dy1, dx1), (dy2, dx2) in pares_direcciones:
144             ny1, nx1 = y + dy1, x + dx1
145             ny2, nx2 = y + dy2, x + dx2
146
147             if (0 <= ny1 < 6 and 0 <= nx1 < 6 and self.tablero[ny1][nx1] in rival and
148                 0 <= ny2 < 6 and 0 <= nx2 < 6 and self.tablero[ny2][nx2] in rival):
149
150                 for cy, cx in [(ny1, nx1), (ny2, nx2)]:
151                     forma = self.tablero[cy][cx] % 4
152                     if jugador == 1:
153                         self.tablero[cy][cx] = 4 + forma
154                     else:
155                         self.tablero[cy][cx] = 8 + forma
156
```

```

157     def empezarJuego(self, nJ):
158         self.victoria = 0
159         self.jugadaAnterior = None
160         self.generarTablero()
161         self.pintarTablero()
162         while self.victoria == 0:
163             valido = False
164             while valido == False:
165                 print("\n\t■ Donuts negros ■")
166                 try:
167                     y = int(input("\tCoordenada vertical: ")) - 1
168                     x = int(input("\tCoordenada horizontal: ")) - 1
169                 except:
170                     y = -1
171                     x = -1
172
173                 valido = self.colocar(1, y, x)
174
175                 if valido == True:
176                     self.jugadaAnterior = (y, x)
177                     self.capturarFichas(1, y, x)
178                     self.pintarTablero()
179                 else:
180                     print("\n\tCasilla no válida.")
181             valido = False
182             if nJ == 1:
183                 validas = self.movimientosValidos()
184                 if validas:
185                     y, x = validas[randrange(len(validas))]
186                     self.colocar(2, y, x)
187                     self.jugadaAnterior = (y, x)
188                     self.capturarFichas(2, y, x)
189                     print("\n\t■ Donuts blancos (IA) ■")
190                     self.pintarTablero()
191                 else:
192                     print("\n\t¡Empate!")
193             break

```

```

194         else:
195             while valido == False:
196                 print("\n\t Donuts blancos ")
197                 try:
198                     y = int(input("\tCoordenada vertical: ")) - 1
199                     x = int(input("\tCoordenada horizontal: ")) - 1
200                 except:
201                     y = -1
202                     x = -1
203                 valido = self.colocar(2, y, x)
204                 if valido == True:
205                     self.jugadaAnterior = (y, x)
206                     self.capturarFichas(2, y, x)
207                     self.pintarTablero()
208                 else:
209                     print("\n\tCasilla no válida.")
210             self.verificarVictoria()
211
212             if self.tableroLleno():
213                 print("\n\t¡Empate!")
214                 break
215             if self.victoria == 1:
216                 print("\n¡Ganan los donuts negros!")
217             if self.victoria == 2:
218                 print("\n¡Ganan los donuts blancos!")
219

```

```

222 class Menu:
223     def __init__(self):
224         self.sel = 0
225         self.juego = Donuts()
226     def iniciar(self):
227         while not (self.sel == 4):
228             print("\n\t Donuts ")
229             print("\n\t1. 1 jugador.")
230             print("\n\t2. 2 jugadores.")
231             print("\n\t3. Leer las reglas.")
232             print("\n\t4. Salir.")
233             try:
234                 self.sel = int(input("\n\tEscribe una opción: "))
235             except:
236                 self.sel = 0
237             match self.sel:
238                 case 1:
239                     self.juego.empezarJuego(1)
240                 case 2:
241                     self.juego.empezarJuego(2)
242                 case 3:
243                     print("\n\tDonuts es un juego similar al 3 en raya.")
244                     print("\tDebes colocar 5 donuts consecutivos en una línea antes que tu oponente.")
245                     print("\tSe genera un tablero nuevo aleatorio cada partida.")
246                     print("\tSolo puedes colocar tu donut en las direcciones marcadas por la línea de la última casilla jugada.")
247                     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")
248                     print("\tSi ninguna de las dos opciones anteriores es posible, puedes colocar tu donut donde quieras.")
249                     print("\tSi colocas un donut entre dos enemigos en línea recta, pasan a ser tuyos.")
250                     print("\tPuedes jugar contra la IA o contra un amigo.")
251                 case 4:
252                     print("\n\tSaliendo del programa...")
253                 case _:
254                     print("\n\tERROR: opción no válida.")
255
256 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:

```