Pregunta 4 (30 %)

Has sido seleccionado como voluntario para los Juegos Olímpicos 2016, y te han asignado como ayudante de los jueces en la prueba de gimnasia. Se te pide que lleves el registro de los puntajes en el "All Around" masculino, y que al final de la ronda entregues los nombres de los dos competidores con puntajes más altos, el competidor con puntaje más bajo y la cantidad de gimnastas que participaron.

Escribe un programa en Python que permita ingresar, consecutivamente, el nombre y puntaje de cada participante hasta que el usuario ingrese "FIN" en el nombre del competidor. Cuando termine el ingreso de participantes, tu programa deberá mostrar los resultados solicitados.

Un ejemplo de diálogo es:

```
>> Nombre de competidor: Tomas Gonzalez
>> Ingrese puntaje: 9.8
>> Nombre de competidor: Louis Smith
>> Ingrese puntaje: 8.3
>> Nombre de competidor: Zou Kai
>> Ingrese puntaje: 7.7
>> Nombre de competidor: Diego Hypolito
>> Ingrese puntaje: 8.5
>> Nombre de competidor: FIN

>> Puntaje maximo: 9.8, Tomas Gonzalez
>> Segundo puntaje maximo: 8.5, Diego Hypolito
>> Puntaje minimo: 7.7, Zou Kai
>> Numero de competidores: 4
```

Nota: asume que siempre se ingresarán los datos de al menos un competidor, y que un mismo competidor no será ingresado dos veces. Solo se ingresarán los datos de los competidores masculinos.

```python
nombre = input("Nombre competidor: ")
competidores = []

while nombre != 'FIN':
    puntaje = float(input('Ingrese puntaje: '))
    competidor = [puntaje, nombre]
    competidores.append(competidor)
    nombre = input("Nombre competidor: ")

competidores.sort(reverse=True)
if len(competidores) > 0:
    print('Puntaje maximo:', competidores[0][0], competidores[0][1])
if len(competidores) > 1:
    print('Segundo puntaje maximo', competidores[1][0], competidores[1][1])
if len(competidores) > 0:
    print('Puntaje minimo', competidores[-1][0], competidores[-1][1])
print('Numero de competidores:', len(competidores))
```

Pregunta 3 (1/4)

La empresa EcoSushi te ha encargado que la ayudes a organizar su proceso de despacho de pedidos, para lo cual utiliza repartidores en bicicleta. Cada pedido tiene el nombre del cliente, una dirección de despacho, una descripción (que indica lo que el cliente pidió) y una coordenada, x e y, asociadas a la dirección del cliente. Existe un despachador que organiza el reparto, el cual recibe los pedidos una vez que están terminados y los distribuye a los repartidores respetando el orden en que estuvieron listos. Para decidir a qué repartidor asignará un pedido, el despachador calcula la distancia entre el pedido que quiere asignar y el último pedido en la lista de cada repartidor, asignándoselo a aquel repartidor con el que el pedido tiene una menor distancia (si hay varios con la misma distancia mínima, se lo puede asignar a cualquiera de ellos). La función de distancia que usa el despachador es:

$$\sqrt{(x_{pedido} - x_{ultimo\_lista\_repartidor})^2 + (y_{pedido} - y_{ultimo\_lista\_repartidor})^2} \quad (1)$$

Para ayudar a la empresa EcoSushi, deberás crear lo siguiente:

a) **(4 puntos)** La clase `Pedido`, con el método `__init__(self, nombre, dirección, x, y, descripción)`.

b) La clase `Repartidor`, con los siguientes métodos:

- **(4 puntos)** `__init__(self, nombre)`, pero que además tiene como atributo la lista de pedidos que debe despachar.
- **(2 puntos)** `agregar_pedido(self,pedido)`, que agrega un pedido al repartidor.
- **(12 puntos)** `distancia_ultimo_pedido(self,pedido)`, que retorna la distancia entre el último pedido de la lista del repartidor y el pedido recibido como parámetro. Cuando no hay ningún pedido en la lista del repartidor este método debe retornar cero.
- **(6 puntos)** `mostrar_ruta(self)`, que muestra la ruta del repartidor imprimiendo una línea por cada pedido, mostrando el nombre del repartidor, el nombre del cliente y la dirección del cliente.

c) La clase `Despachador`, con los siguientes métodos:

- **(4 puntos)** `__init__(self)`, que tiene como atributos la lista de pedidos que quiere despachar y la lista de repartidores.
- **(2 puntos)** `agregar_pedido(self,pedido)`. Este método solamente agrega un pedido al despachador, aún no lo asigna a los repartidores.
- **(2 puntos)** `agregar_repartidor(self,repartidor)`, que agrega un repartidor.
- **(24 puntos)** `despachar(self)`, que distribuye todos los pedidos de la lista entre los repartidores utilizando las reglas indicadas anteriormente, utilizando el método `distancia_ultimo_pedido` de la clase Repartidor para tomar la decisión de a quién asignar el pedido. Cuando los pedidos han sido asignados a los repartidores, la lista de pedidos debe quedar vacía (sin pedidos por asignar).

```python
class Pedido:
    def __init__(self, nombre, direccion, x, y, descripcion):
        self.nombre = nombre
        self.direccion = direccion
        self.x = x
        self.y = y
        self.descripcion = descripcion
    def __str__(self):
        return self.nombre + ' ' + self.direccion
```

```python
class Repartidor:
    def __init__(self, nombre):
        self.nombre = nombre
        self.pedidos = []

    def agregar_pedido(self, pedido):
        self.pedidos.append(pedido)
    def distancia_ultimo_pedido(self, pedido):
        if len(self.pedidos) == 0:
            return 0
        else:
            ult = self.pedidos[-1]
            dist = (pedido.x - ult.x) ** 2 + (pedido.y - ult.y) ** 2
            dist = dist ** 0.5
            return dist

    def mostrar_ruta(self):
        for pedido in self.pedidos:
            print(self.nombre, pedido)
```

```python
class Despachador:
    def __init__(self):
        self.pedidos = []
        self.repartidores = []
    def agregar_pedido(self, pedido):
        self.pedidos.append(pedido)

    def agregar_repartidor(self, repartidor):
        self.repartidores.append(repartidor)
```

```python

    def despachar(self):
        while self.pedidos:
            pedido = self.pedidos.pop(0)
            min_repartidor = None
            for repartidor in self.repartidores:
                if min_repartidor is None:
                    min_repartidor = repartidor
                else:
                    dist = repartidor.distancia_ultimo_pedido(pedido)
                    dist_min = min_repartidor.distancia_ultimo_pedido(pedido)
                    if dist < dist_min:
                        min_repartidor = repartidor
            min_repartidor.agregar_pedido(pedido)
```