


## Python - Examen: Tratamiento de excepciones (jueves - 10:30)

 **Disponible desde:** jueves, 23 de febrero de 2023, 10:50

 **Límite de entrega:** jueves, 23 de febrero de 2023, 11:30

 **Ficheros requeridos:** main.py, solution.py ( [Descargar](#))

 **Número máximo de ficheros:** 3

**Tipo de trabajo:**  Individual

Se quiere desarrollar una función llamada **is\_upper\_triangular\_matrix** que devuelva si una matriz es triangular superior o no. Una matriz triangular superior se define como aquella matriz cuadrada que sus elementos bajo la diagonal principal son ceros. La matriz se pasa a la función en forma de lista de listas.

Desarrolle la función **is\_upper\_triangular\_matrix** teniendo en cuenta que, además de devolver verdadero o falso, lanzará la excepción **BadMatrix** con el texto "Matriz no cuadrada" si la matriz pasada no es cuadrada y lanzará la excepción **BadMatrix** con el texto "Dato no convertible a entero" en caso de que un elemento no sea int ni ristra (str), o siendo una ristra (str) no sea posible obtener un entero a partir de la ristra para considerarlo.

```
triangular_superior= [
    [1, 0, 0, "1"]
    [0, 4, 0, 0],
    [0, 0, "-2", 0],
    [0, 0, "0", -1]
]
```

## Ficheros requeridos

main.py

```
1  """Ejemplo de uso de la función requerida."""
2
3  from solution import BadMatrix
4  from solution import is_upper_triangular_matrix
5  m1 = [[1, -1, 2, 3], [0, 4, 0, 0], [0, 0, -2, 0], [0, 0, 0, -1]]
6  m2 = [[1, 0, 0, 0], [0, 4, 0, 0], [0, 0, -2, 0], [0, 0, 0, -1]]
7  m3 = [[1, 0, 0, 0], [0, 4, 0, 0], [0, 0, -2, 0], [0, 0, 0, -1], [0]]
8  m4 = [[1, 0, 0, 0], [0, 4, 0, 0], [0, 0, -2, 0], [0, 0, 0]]
9  m5 = ["1", 0, 0], ["0", "44", 1], [0, "-0", "341"]
10 m6 = [[[]], 0, 0], [0, 44, 1], [0, -0, 341]]
11 m7 = [[4, 0, 0], [0, 44, 1], [0, -0, "hola"]]
12 for m in [m1, m2, m3, m4, m5, m6, m7]:
13     try:
14         print(is_upper_triangular_matrix(m))
15     except BadMatrix as err:
16         print("Error en matriz", err)
17         print(m)
18
19
20
```

solution.py

```
1 """Module with the answer to the problem."""
2
3
4 def is_upper_triangular_matrix(matrix):
5     """Return True if rows is an upper triangular matrix."""
6     pass
7
8 # NO MODIFIQUE EL CODIGO DEBAJO DE ESTA LINEA
9
10
11 class BadMatrix(Exception):
12     """Exception for BadMatrix."""
13
14     pass
15
```

## Python - Examen: Crear clases básico (jueves - 10:30)

 **Disponible desde:** jueves, 2 de marzo de 2023, 10:45

 **Límite de entrega:** jueves, 2 de marzo de 2023, 12:00

 **Ficheros requeridos:** main.py, solution.py ( [Descargar](#))

**Tipo de trabajo:**  Individual

Se desea crear una clase llamada **Matriz** para representar objetos de tipo matriz. Los métodos que suministrarán los objetos de esta clase serán los siguientes:

- El constructor que pasándole una lista de listas como datos de la matriz posibilitará inicializar un objeto.
- **es\_cuadrada** que devolverá verdadero si la matriz tiene el mismo número de filas que de columnas en todas sus filas.
- **nfilas** que devolverá el número de filas de la matriz.
- **ncolumnas** que devolverá el número máximo de columnas en las filas de la matriz
- **get\_str** devuelve una ristra que representa la matriz de forma que si se imprime la ristra la matriz se muestre en líneas. Cada línea será el resultado de la función str de cada fila de la matriz.

Ejemplo:

```
Para la matriz [[1, 0, 0, 0], [0, 4, 0, 0], [0, 0, -2, 0], [0, 0, 0, -1], [0]]
es_cuadrada => False
nfilas => 5
ncolumnas => 4
get_str => [1, 0, 0, 0]
[0, 4, 0, 0]
[0, 0, -2, 0]
[0, 0, 0, -1]
[0]
```

## Ficheros requeridos

main.py

```
1  """Ejemplo de uso de la función requerida."""
2
3  from solution import Matriz
4
5  ms = []
6  ms.append([[1, 1], [0, 0]])
7  # Descomente líneas para probar más casos
8  # ms.append([[1, -1, 2, 3], [0, 4, 0, 0], [0, 0, -2, 0], [0, 0, 0, -1]])
9  # ms.append([[1, 0, 0, 0], [0, 4, 0, 0], [0, 0, -2, 0], [0, 0, 0, -1]])
10 # ms.append([[1, 0, 0, 0], [0, 4, 0, 0], [0, 0, -2, 0], [0, 0, 0, -1], [0]])
11 # ms.append([[1, 0, 0, 0], [0, 4, 0, 0], [0, 0, -2, 0], [0, 0, 0]])
12 # ms.append(["1", 0, 0], ["0", "44", 1], [0, "-0", "341"])
13 # ms.append([[[]], 0, 0], [0, 44, 1], [0, -0, 341])
14 # ms.append([[4, 0, 0], [0, 44, 1], [0, -0, "hola"]])
15
16 for i, m in enumerate(ms):
17     mat = Matriz(m)
18     print(f"===== MATRIZ {i+1} =====")
19     print("es_cuadrada", mat.es_cuadrada())
20     print("nfilas", mat.nfilas())
21     print("ncolumnas", mat.ncolumnas())
22     print("Representación ", mat.get_str())
23
```

solution.py

```
1 | ""Module with the answer to the problem.""
2
```

## Python - Examen: Propiedades (Jueves - 10:30)

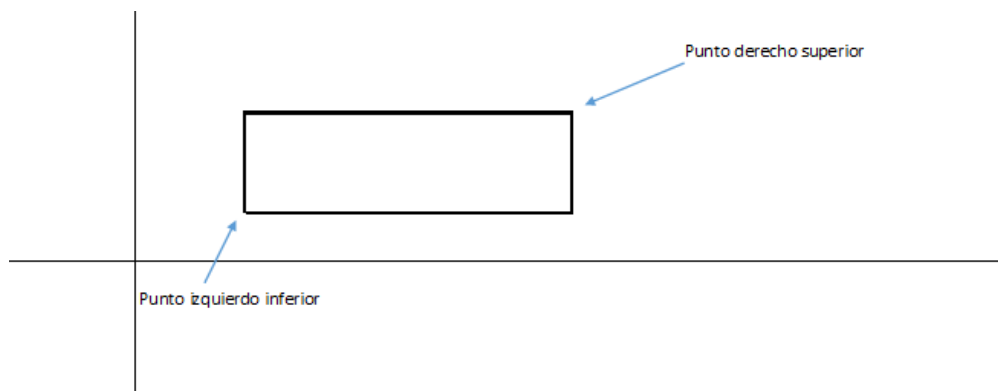
📅 **Disponible desde:** jueves, 9 de marzo de 2023, 10:40

📅 **Límite de entrega:** jueves, 9 de marzo de 2023, 12:20

📁 **Ficheros requeridos:** main.py, solution.py (📄 [Descargar](#))

**Tipo de trabajo:** 👤 Individual

Se desea crear una clase llamada **Rectangulo** para representar rectángulos en un plano de dos dimensiones. Los lados de los rectángulos a representar son paralelos a los ejes x e y. Para representar los rectángulos se parte de los dos puntos que representan su límite inferior izquierdo y el límite superior derecho.



Se dispone, ya desarrollada, de una clase llamada **Punto** que permite almacenar la coordenada x e y de cada punto (Ver código suministrado para detalles de uso). Usando apropiadamente la clase Punto, desarrolle la clase **Rectangulo** para que suministren las siguientes características:

- El inicializador (constructor) que pasándole dos puntos como parámetros establece el rectángulo a crear. Se pasa como primer parámetro el punto de la esquina inferior izquierda y como segundo el de la esquina derecha superior. En caso de que se pasen puntos que no formen un rectángulo al no ser el de la esquinas correspondientes e, método debe lanzar la excepción *BadRectangle* establecida en el módulo *solution*.
- **perimetro** propiedad de solo lectura que representa el perímetro del rectángulo.
- **superficie** propiedad de solo lectura que representa la superficie del rectángulo.
- **esta\_contenido** que pasándole un objeto de tipo Punto devuelve si está en el rectángulo o no.
- **ini** propiedad de tipo Punto de lectura y escritura que representa el punto inferior izquierdo. En caso de que se pasé un punto que con el otro no forme un rectángulo se debe lanzar la excepción *BadRectangle*.
- **fin** propiedad de tipo Punto de lectura y escritura que representa el punto superior derecho. En caso de que se pasé un punto que con el otro no forme un rectángulo se debe lanzar la excepción *BadRectangle*.
- La representación informal del rectángulo será una ristra en la que aparezcan entre "<" y ">" una representación de los Puntos de las dos esquinas indicadas separados por ", " coma y espacio. Cada punto se representará como los valores de x e y entre paréntesis y separados por ", " coma y espacio:

Ejemplo: para Rectangulo(Punto(1, 2), Punto(3, 4)) se representará como "<(1, 2), (3, 4)>"

```
Puntos (2, 3) y (3, 4)
Perímetro 4
Superficie 1
Representación <(2, 3), (3, 4)>
Después de escalar 1.5
Perímetro 6.0
Superficie 2.25
Representación <(2, 3), (3.5, 4.5)>
```

# Ficheros requeridos


main.py

```
1 """Ejemplo de uso de la funciones requerida."""
2
3 from solution import Rectangulo, Punto, BadRectangle
4 import inspect
5
6 tests = [
7     (Punto(2, 3), Punto(3, 4)),
8     (Punto(-2, 3), Punto(0, 5)),
9     (Punto(0, 5), Punto(-2, 6)),
10    (Punto(-2, 3), Punto(0, 2)),
11 ]
12
13 for p1, p2 in tests:
14     try:
15         print("Puntos", p1.get_str(), p2.get_str())
16         r = Rectangulo(p1, p2)
17         print("Perímetro", r.perimetro)
18         print("Superficie", r.superficie)
19         print("Está contenido el Punto(0, 0)", r.esta_contenido(Punto(0, 0)))
20         print("Rectángulo", r)
21     except BadRectangle as err:
22         print(f"Rectángulo erróneo {p1.get_str()} {p2.get_str()}")
23
24
```

solution.py

```
1 """Module with the answer to the problem."""
2
3 # AÑADA AQUÍ LA CLASE Rectangulo
4
5 # NO MODIFIQUE EL CODIGO DEBAJO DE ESTA LINEA
6
7
8 class Punto:
9     """Punto con dos propiedades x e y."""
10
11     def __init__(self, x, y):
12         """Inicializa punto."""
13         self.x = x
14         self.y = y
15
16     def get_str(self):
17         """Devuelve punto como str."""
18         return f"({self.x}, {self.y})"
19
20
21 class BadRectangle(Exception):
22     """Exception for BadRectangle."""
23
24     pass
25
```

## Python - Examen: Variables/métodos de clase (Jueves - 10:30)

 **Disponible desde:** jueves, 16 de marzo de 2023, 10:40

 **Límite de entrega:** jueves, 16 de marzo de 2023, 11:40

 **Ficheros requeridos:** solution.py ( [Descargar](#))

**Tipo de trabajo:**  Individual

Realice las siguientes acciones:

1. Añada a la clase `ColorRGB` una variable de clase llamada ***nombres*** que sea un *diccionario*. Las claves del diccionario serán tuplas que representan una combinación rgb y las contraclaves serán strings representando nombres de colores conocidos. Este diccionario estará inicialmente vacío.
2. Añada a la clase `ColorRGB` un método de clase ***establece\_nombre(cls, color, nombre)***, que servirá para añadir un nuevo ítem a la variable de clase *nombres*. El parámetro *color*, que se usará como clave del diccionario *nombres*, será una tupla de tres números enteros con valores en el rango de 0 a 255, y el parámetro *nombre*, que se usará como contraclave, será una string.  
El método debe asegurarse de que el parámetro *color* sea una tupla de longitud 3 y que el parámetro *nombre* sea una string. En caso de no cumplirse la primera condición se lanzará la excepción `TypeError` con el mensaje "*color*" y, si se cumple la primera pero no la segunda, se lanzará la excepción `TypeError` con el mensaje "*nombre*". No es necesario comprobar que los elementos de la tupla sean de tipo *int*.
3. Añada a la clase `ColorRGB` un propiedad de solo lectura llamada *nombre* que de el nombre del color representado por el objeto *self*. Para ello accederá al diccionario *nombres* usando como clave una tupla formada por la combinación de valores (red, green, blue) del objeto *self*. En caso de no encontrarse en *nombres* un ítem con dicha combinación, el método *nombre* devolverá el valor "desconocido".

## Ficheros requeridos

solution.py

```
1  """Clase a modificar para el ejercicio."""
2
3
4  class ColorRGB:
5      """Representa un color en formato RGB."""
6
7      def __init__(self, red, green, blue):
8          """Inicializa un objetos con valores para red, green y blue."""
9          self.red = red
10         self.green = green
11         self.blue = blue
12
13         def __str__(self):
14             """Representación informal colorRGB."""
15             color = (self.red, self.green, self. blue)
16             return f"{color}"
17
18
19  if __name__ == "__main__":
20      c = ColorRGB(0, 0, 0)
21      print(c)
22      print(c.nombre)
23
```

[VPL](#)

## Python - Examen: Herencia (jueves - 10:30)

📅 **Disponible desde:** jueves, 23 de marzo de 2023, 11:00

📅 **Límite de entrega:** jueves, 23 de marzo de 2023, 11:30

📁 **Ficheros requeridos:** solution.py (📄 [Descargar](#))

**Tipo de trabajo:** 👤 Individual

El módulo *clases* oferta una clase llamada *Barco* que tiene dos propiedades llamadas *manga* y *eslora* que se inicializan al crear un objeto de la clase *Barco*.

1. Implemente, en el archivo solution.py, una nueva clase llamada **Velero**, que herede de la clase *Barco*.

El inicializador de la clase **Velero** tendrá, aparte de los parámetros necesarios para la expresión constructora de los objetos de la clase *Barco*, un parámetro adicional para inicializar una nueva propiedad, que deberá añadirse a la clase, llamada **palos** (indicará el número de mástiles del velero)

2. Incluya en la clase **Velero** el código mínimo que sea necesario para que dos objetos de esa clase se puedan comparar

Dos objetos de la clase **Velero** se comparan en función de las propiedades heredadas de *Barco*: dados dos objetos de la clase *Velero*, es menor aquel para el que el producto de multiplicar *manga por eslora es menor*. Si este *propducto es igual*, ambos objetos son iguales

## Ficheros requeridos

solution.py

```
1 # Escriba aquí su código
2
3
```