Lista 04 - Prog 01 2024

Emilio Vital Brazil

Entrega: 20 de maio 2024

Todas as questões abaixo serão consideradas usando a sintaxes **Python 3.10**.

1. Observe o seguinte codigo e escreva a saida padrão esperada para cada item abaixo:

```python
x = 2
myList = [1, 2, 3, 4]
myDict = {
    'chave1': 1,
    'chave2': x,
    3       : 'Value',
    (1,2)   : 'Nossa que massa!!!',
    'L'     : myList
}
x = 3
myDict['L'][2] = 5
```

(a)

```python
print(myDict['chave1'])
print(myDict['chave2'])
```

(b)

```python
print(myDict[3]+' is bad')
```

(c)

```python
print(myDict[(1,2)][6:16])
```

(d)

```python
print(myList)
```

2. Observe o seguinte código e a sua saída de erro padrão:

```
class Vector():
    def __ints__(self, values):
        """
        Initializes the vector with the given values.
        """
        self.values = values
        self.dim    = len(self.values)

    def add( self, other_vector):
        """
        Returns a new vector that represents the sum of this
        vector and the given vector.
        It checks if the dimensions of the two vectors are equal,
        if not returns an empty list.
        """
        if other_vector.dim != self.dim:
            return []
        new_vector = [0] * self.dim
        for i in range(self.dim):
            new_vector[i] = self.values[i] + self.values[i]
        return new_vector

v = Vector([0.0 , 2.0 , 3.5 ])
u = Vector([1.0, 0.5])
w = Vector([2.0, 1.5])
print(v.add(u))
print(u.add(w))
```

```
TypeError  Traceback (most recent call last)
Cell In[9], line 23
     20             new_vector[i] = self.values[i] + self.values[i]
     21         return new_vector
---> 23 v = Vector([0.0 , 2.0 , 3.5 ])
     24 u = Vector([1.0, 0.5])
     25 w = Vector([2.0, 1.5])

TypeError: Vector() takes no arguments
```

(a) Qual mudança precisa ser feita para o codigo rodar sem erro?

(b) Qual saida padrão se for resolvido o erro de 'TypeError'?

(c) A saida padrão está diacordo com o esperado? Qual o erro de lógica que você indicaria dado o comentário do metódo *add*?

3. Escreva um código que:

   (a) crie uma classe *Circle*, que receba um ponto (x, y) e um *float* como argumentos, e seja capaz de responder se o ponto está dentro, fora ou sobre a circunferência do círculo.

   (b) crie uma classe *Line_Segment*, que é construído por dois pontos. Esse objeto deve ter um metodo que responde se um ponto (x, y) pertence ou não ao segmento.

4. Crie uma função que recebe dois objetos um do tipo *Circle* e outro do tipo *Line_Segment* (do item anterior) e retorna **True** se um segmento está totalmente contido no disco aberto definido pela circunferência do objeto *Circle*, e **False** caso contrario.

5. O produto diádico ($\otimes$) é uma operação matemática importante em álgebra linear que envolve o produto tensorial de dois vetores. O resultado dessa operação é uma matriz que representa a relação entre os dois vetores. Por exemplo, sejam $u \in \mathbb{R}^n$ e $v \in \mathbb{R}^m$ tal que $\mathbf{W} = u \otimes v$, $\mathbf{W} \in \mathbb{R}^{nm}$, sendo as entradas de $\mathbf{W}$, $w_{ij} = u_i v_j$.

   Escreva um codigo que tenha uma classe *Vector* que é construida apartir de uma lista de floats. E além do construtor tenha um metodo *dyadic_product* para calcular o produto diádico com outro vetor.

   Por exemplo:

```
>>> u = Vector([0.0, 2.0, 3.5])
>>> v = Vector([1.0, 0.5])
>>> u.dyadic_product(v)
[[0.0, 0.0], [2.0, 1.0], [3.5, 1.75]]
>>> v.dyadic_product(u)
[[0.0, 2.0, 3.5], [0.0, 1.0, 1.75]]
```