8 Aula 08: 27/AGO/2019

8.1 Aulas passadas

- POO: classes `Fracao`, `Complexo`, `Polinomio`
- Pilhas: sequências bem formadas e notação polonesa.
- Array2D e introdução ao `Numpy.ndarray`
  - criação usando lista e `np.full( shape, val )`
  - fatias versos vistas
  - atributos: `shape`, `size`, `ndim`, `dtype`

8.2 Hoje

O que você deve saber para usar arrays

- Operações comuns (`+`, `*`, `-`, `/`) são realizadas
- elemento a elemento para `arrays` de mesmo `shape`
- para todos os elementos usando um escalar
- use `@` para multiplicar matrizes
- funções universais como `sqrt()`, `max()` e `min()`
- *broadcasting*, hmm, dexamos de lado
- vista de um `array`
- vistas não são cópias
- dados podem ser compartilhados por várias vistas
- implementando `reshape`, vistas, atributo `T` método `transpose()`
- `copy()`: para clonar
- manipulação de vistas usando pedaços de array: `fliplr()` e `flipud()`

Os recursos de `np,arrays` usados no EP04 foram:

```
if type(valor) == np.ndarray:
self.data = np.full((nlins, ncols), valor)
return self.data.shape
novo_data = np.copy(self.data[tlin:blin, tcol:bcol])
soma = self.data + other.data # adição de ndarrays
prod = self.data * alfa       # multiplicação de ndarrays
self.data = self.data.T       # transposição
self.data = np.reshape(self.data, (nlin, ncol))
self.data = np.flipud(self.data)
self.data = np.fliplr(self.data)
```

8.3 Exercício 0: Comportamento geral de funções numéricas comuns

O que você espera na saída para operações com arrays de mesmo tamanho e com escalares:

```
>>> import numpy as np

>>> a = np.full( (3,2), 1)
>>> b = np.array( [[1,2], [3,4], [5,6]])
>>> a + b
array([[2, 3],
       [4, 5],
       [6, 7]])
>>> b + 1
array([[2, 3],
       [4, 5],
       [6, 7]])
>>> b/(a*2)
array([[0.5, 1. ],
       [1.5, 2. ],
       [2.5, 3. ]])

>>> b**2
array([[ 1,  4],
       [ 9, 16],
       [25, 36]])
>>> b.max()
6
>>> np.sqrt(b)
array([[1.        , 1.41421356],
       [1.73205081, 2.        ],
       [2.23606798, 2.44948974]])
```

8.4 Exercício 1: Vistas x cópias

Escreva o método `reshape( newshape )`, que retorna uma vista dos mesmos dados com um novo shape.

```
>>> a = Array2D( (2,3), 1)
>>> print(a)
1 1 1
1 1 1
>>> b = a.reshape(3,2)
>>> print(b)
1 1
1 1
1 1
>>> c = b.reshape(1,6).copy()
>>> print(c)
1 1 1 1 1 1
>>> c.data[2] = 0
>>> print(b)
```

```
1 1
1 1
1 1
>>> b.data[2] = 0
>>> print(a)
1 1 0
1 1 1
```

Usando a Classe `Array2D` da aula passada, escreva o método `reshape()` e `copy()`.

```
def reshape(self, shape ):
    vista = Array2D( (0,0), 0)  # array de tamanho zero
    nl, nc = shape
    vista.shape = shape
    vista.size  = nl * nc
    vista.dtype = self.dtype
    vista.data = self.data   # compartilha os mesmos dados!!
    return vista

def copy(self):
    copia = Array2D( self.shape, self.dtype(0) ) # array de mesmo tamanho e tipo

    # errado
    copia.data = self.data # errado !!!

    # correto
    for i in self.size:
        copia.data[i] = self.data[i]
    return copia
```

8.5 Exercício 2: FlipH e FlipV

OBS: Começar com FlipV que parece mais fácil por trocar linhas, ou fazer só o flip horizontal para mostrar a troca de colunas.

Escreva uma função `fliplr` que recebe um array 2D e altera o array espelhado-o horizontalmente, ou seja, as colunas à esquerda são trocadas com as da direita.

```
import numpy as np
def main():
    a = np.array( [1,2,3,4] ).reshape(2,2)
    print(a)
    espalha_ed(a)
    print(a)

def fliplr( ar ):
    ''' (array) -> array
    espelha as colunas.
    '''
    nlin, ncol = ar.shape
    meio = ncol // 2
```

```
        for i in range(meio):
            fim = nlin-1-i
            ar[:,i], ar[:,fim] = ar[:,fim], ar[:,i]
    main()
```

OBS: `np.fliplr(array)` e `np.flipup(array)` devolvem vistas do array.