1 Aula 07: 22/AGO/2019

1.1 Aulas passadas

Pilhas: sequências bem formadas e notação polonesa.

1.2 Hoje

Vamos falar de `arrays` usando `Numpy` devido grande utilidade em computação científica e engenharia e dados.

Os recursos de `arrays` usados no EP04 são:

```
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 arrays
prod = self.data * alfa        # multiplicação de arrays
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)
self.data = np.rot90(self.data, 3)
data = self.data.astype('float')
data /= data.max()/255.0
self.data = data.astype('int32')
```

Com isso, na aula de hoje, as alunas e alunos acabarão vendo:

- Criação de `arrays`: usando uma lista, `zeros()`, `ones()`, `full()`
- Principais atributos: `ndim`, `shape` (formato), `size`, `ndim`
- Acesso:
  - 1 dimensional: muito parecido com listas, fatia `[ ini : fim : passo ]`, fechado aberto, índice negativo
  - `n` dimensional: usa `tuple` entre colchetes ao invés um par por dimensão
  - fatia nD: dá para pegar linha, coluna e blocos
  - **CUIDADO**: fatias devolvem *vistas* e não são *clones* do `array`.
  - use método `copy()` se for necessário clonar

- Operações básicas com arrays:
  - soma e produto de dois arrays de mesmo formato
  - soma e produto de array com escalar
- Manipulação do formato dos arrays:
  - `reshape()` versus `resize()`
  - `ravel()`
  - `T`

1.3 Problema

Escreva uma função que recebe uma matriz `A` com `nlins` linhas e `ncols` colunas, e dois índices `lin` e `col` de uma linha e coluna de `A`, e retorna duas listas, a primeira com os elementos da linha `lin` e a segunda com os elementos da coluna `col`.

1.4 `Array` e o módulo `NumPy`

Em computação científica é muito comum o uso de arrays. Arrays são estruturas de dados semelhantes às listas do Python, mas não tão flexíveis. Em um array todos os elementos devem ser de um mesmo tipo, tipicamente numérico, como int ou float. Além disso, o tamanho de um array não pode ser modificado, ao contrário de listas que podem crescer dinamicamente. Em contrapartida, o uso de arrays é muito mais eficiente e facilita a computação de grandes volumes de dados numéricos. Isso faz com que arrays sejam particularmente úteis em computação científica.

Para trabalhar com `array`s em Python vamos utilizar o módulo `NumPy` (=*Numeric Python*). O `NumPy` define a classe `ndarray` para encapsular `array`s de `n{ dimensões. Os arrays do`NumPy`tem tamanho fixo, assim, para se mudar o tamanho de um`ndarray`, cria-se um novo`array‘ e o original é removido.

`NumPy` oferece várias funções *pré-compiladas* que tornam o processamento de `array`s mais eficiente. Aqui vamos introduzir apenas um pequeno conjunto desses recursos. Para saber mais, consulte a documentação do módulo NumPy.

1.5 Conceitos de NumPy

De 2.1 Até 2.4 do guia do usuário.

Um array em NumPy é uma tabela multidimensional de elementos do mesmo tipo, indexados por uma tupla de inteiros positivos. As dimensões são chamadas de `eixos` (axes).

A classe básica é o `ndarray` que também tem o apelido de `array‘. Note que o`numpy.array`é diferente do tipo nativo do Python`array.array“ que só manipula arrays de uma dimensão e apresenta menos funcionalidades.

1.6 Principais atributos da `ndarray`

- `ndarray.ndim`: número de eixos
- `ndarray.shape`: dimensões do array
- `ndarray.size`: número total de elementos
- `ndarray.dtype`:
- `ndarray.itemsize`:
- `ndarray.data`:

1.7 `arrays` a partir de listas

```
>>> import numpy as np
>>> a = [[1,2,3], [6,5,4]]
>>> b = np.array(a)
```

```
>>> print(a)
[[1, 2, 3], [6, 5, 4]]
>>> print(b)
[[1 2 3]
 [6 5 4]]
>>> type(b)
<class 'numpy.ndarray'>
>>> type(a)
<class 'list'>
>>> c = np.array(a, float)
>>> print(c)
[[ 1.  2.  3.]
 [ 6.  5.  4.]]
```

1.8 Dimensão de um array

```
# M é um mapa inicial, com moldura (zeros em toda a volta)
mapa = [[0,0,0,0,0,0],
        [0,0,0,1,0,0],
        [0,1,0,1,0,0],
        [0,0,1,1,0,0],
        [0,0,0,0,0,0],
        [0,0,0,0,0,0]]

mapa = np.array(mapa)

nlins = len(mapa)
ncols = len(mapa[0])

nlins, ncols = mapa.shape
```

1.9 arrays com zeros, ou uns, ou identidade

```
   # cria matriz de vizinhos com zeros
   viz = []
   for lin in range(nlins):
       viz.append([0] * ncols)

   viz = np.zeros((nlins,ncols), int)
```

```
    >>> z = np.zeros( (2,3) ) # dimensao 2x3
    >>> u = np.ones( (3,2), int)
    >>> I = np.identity( 2, float)
    >>> print(z)
    [[ 0.  0.  0.]
    [ 0.  0.  0.]]
    >>> print(u)
    [[1 1]
```

```
    [1 1]
    [1 1]]
    >>> print(I)
    [[ 1.  0.]
    [ 0.  1.]]
    >>>
```

1.10 Índices e fatias de arrays

```
>>> import numpy as np
>>> a = list(range(1,10))
>>> a
[1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> a[1:5]
[2, 3, 4, 5]
>>> a[1:7:2]
[2, 4, 6]
>>> a = np.array(a)
>>> a
array([1, 2, 3, 4, 5, 6, 7, 8, 9])
>>> a[2]
3
>>> a[2:7:2]
array([3, 5, 7])
>>> a[2,7,2]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: too many indices for array
>>> a[[2,7,2]]
array([3, 8, 3])
>>>
```

Ou seja, passando uma lista de indices como índice de array, temos um array com os elementos indexados.

1.11 Caso 2D

```
>>> y = np.array(range(35)).reshape(5,7)
>>> y
array([[ 0,  1,  2,  3,  4,  5,  6],
       [ 7,  8,  9, 10, 11, 12, 13],
       [14, 15, 16, 17, 18, 19, 20],
       [21, 22, 23, 24, 25, 26, 27],
       [28, 29, 30, 31, 32, 33, 34]])
>>> y[1:5:2]
array([[ 7,  8,  9, 10, 11, 12, 13],
       [21, 22, 23, 24, 25, 26, 27]])
>>> y[1:5:2,::3]
array([[ 7, 10, 13],
```

```
       [21, 24, 27]])
>>> y[1:5:2][::3]
array([[ 7,  8,  9, 10, 11, 12, 13]])
>>> y[[1,3,5],[2,3,4]]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: index 5 is out of bounds for axis 0 with size 5
>>> y[[1,3,4],[2,3,4]]
array([ 9, 24, 32])
```

Embora seja possível usar a mesma forma para acessar elementos de um array, uma forma mais eficiente é separando os valores por vírgulas, como a seguir:

```
>>> a = [[0, 1, 2, 3, 4], [5, 6, 7, 8, 9]]
>>> a
[[0, 1, 2, 3, 4], [5, 6, 7, 8, 9]]
>>> a[0,0]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: list indices must be integers, not tuple
>>> a[0][0]
0
>>> b = np.array(a)
>>> b[1][2]
7
>>> b[1,3]
8
>>>
```

Em listas não é permitodp o uso de vírgulas para acessar um elemento.

Além de ser mais eficiente, o uso de vírgulas permite formas de fatiamento mais flexíveis, como mostrado nos exemplos a seguir.

```
>>> lista = [[0, 1, 2, 3], [4, 5, 6, 7], [8, 9, 10, 11]]
>>> lista
[[0, 1, 2, 3], [4, 5, 6, 7], [8, 9, 10, 11]]
>>> lista[1:-1]
[[4, 5, 6, 7]]
>>> mat = np.array(lista)
>>> mat
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
>>> mat[1:-1]
array([[4, 5, 6, 7]])
>>> mat[:,2]
array([ 2,  6, 10])
>>> mat[:,2:]
array([[ 2,  3],
       [ 6,  7],
       [10, 11]])
```

```
>>>
```

Observe que a variável lista faz referência a uma matriz de dimensão `(3,4)` e que a fatia `lista[1:-1]` é a matriz contendo apenas a segunda linha. Lembre-se que é possível usar índices negativos para indicar os elementos da direita para a esquerda, e portanto o último elemento pode ser indicado pelo índice -1, o penúltimo por -2, etc.

Quando a lista é transformada para array, o mesmo resultado é obtido usando o fatiamento `mat[1:-1]`. Observe no entanto que, usando vírgulas, podemos selecionar todas as linhas de mat e a coluna de índice 2 usando `mat[:,2]` (= coluna de índice 2) e ainda criar uma matriz contendo as duas últimas colunas usando `mat[:,2:]`.

1.12 Operações com arrays

Apesar da semelhança do tipo ndarrays do NumPy com as listas de Python para o tratamento de índices e fatias, o NumPy oferece muito mais recursos que listas para operação e manipulação de arrays que facilitam a computação científica.

Para arrays com o mesmo número de elementos e forma, os símbolos das operações básicas como *, +, etc, podem ser utilizadas para calcular um novo array "por elemento".

```
>>> import numpy as np
>>> a = np.array([[1,2],[3,4]])
>>> a
array([[1, 2],
       [3, 4]])
>>> b = np.array([[0,2],[3,1]])
>>> b
array([[0, 2],
       [3, 1]])
>>> a+b
array([[1, 4],
       [6, 5]])
>>> a*b
array([[0, 4],
       [9, 4]])
>>> b/a
array([[ 0.  ,  1.  ],
       [ 1.  ,  0.25]])

viz[1:-1,1:-1] += (mapa[0:-2,0:-2] +
                   mapa[0:-2,1:-1] +
                   mapa[0:-2,2:] +
                   mapa[1:-1,0:-2] +
                   mapa[1:-1,2:] +
                   mapa[2:  ,0:-2] +
                   mapa[2:  ,1:-1] +
                   mapa[2:  ,2:])
>>> viz
array([[1, 3, 1, 2],
       [1, 5, 3, 3],
```

```
       [2, 3, 2, 2],
       [1, 2, 2, 1]])
>>> viz > 3
array([[False, False, False, False],
       [False,  True, False, False],
       [False, False, False, False],
       [False, False, False, False]], dtype=bool)
>>>

c = []
for i in range(len(a)):
    c.append(a[i]*b[i])

>>> a = [1,2,3]
>>> b = [4,5,6]
>>> a*b
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: can't multiply sequence by non-int of type 'list'
```

Com numpy:

```
>>> a_array = np.array(a)
>>> b_array = np.array(b)
>>> a_array * b_array
array([ 4, 10, 18])
>>>

Python 3.4.3 |Anaconda 2.3.0 (64-bit)| (default, Jun  4 2015, 15:29:08)
[GCC 4.4.7 20120313 (Red Hat 4.4.7-1)] on linux
Type "copyright", "credits" or "license()" for more information.
>>> import numpy as np
>>> mapa = np.array(
    [[0,0,0,0,0,0],
     [0,0,0,1,0,0],
     [0,1,0,1,0,0],
     [0,0,1,1,0,0],
     [0,0,0,0,0,0],
     [0,0,0,0,0,0]]
)
>>> mapa
array([[0, 0, 0, 0, 0, 0],
       [0, 0, 0, 1, 0, 0],
       [0, 1, 0, 1, 0, 0],
       [0, 0, 1, 1, 0, 0],
       [0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0]])
>>> mapa[0:-2,0:-2]
array([[0, 0, 0, 0],
       [0, 0, 0, 1],
       [0, 1, 0, 1],
```

```
       [0, 0, 1, 1]])
>>> (mapa[0:-2,0:-2] + mapa[0:-2,1:-1] + mapa[0:-2,2:] +
                       mapa[1:-1,0:-2] +                   mapa[1:-1,2:] +
                       mapa[2:  ,0:-2] + mapa[2:  ,1:-1] + mapa[2:  ,2:])
array([[1, 3, 1, 2],
       [1, 5, 3, 3],
       [2, 3, 2, 2],
       [1, 2, 2, 1]])
>>> mapa[0:-2,1:-1]
array([[0, 0, 0, 0],
       [0, 0, 1, 0],
       [1, 0, 1, 0],
       [0, 1, 1, 0]])
>>> mapa[0:-2,0:-2]
array([[0, 0, 0, 0],
       [0, 0, 0, 1],
       [0, 1, 0, 1],
       [0, 0, 1, 1]])
>>> mapa[0:-2,2:]
array([[0, 0, 0, 0],
       [0, 1, 0, 0],
       [0, 1, 0, 0],
       [1, 1, 0, 0]])
>>>
```