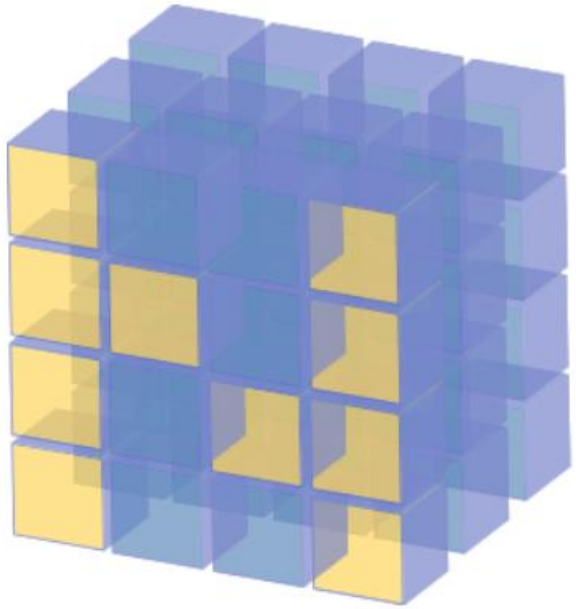
ARTIFICIAL INTELLIGENCE

NUMPY WITH PYTHON

Edit & Colect: Ngo Kien Hoang
Emai: hoang.ngo000187@hcmut.edu.vn
hoang.ngo000187@gmail.com

Numpy (viết tắt của Numerical Python) là một thư viện lõi phục vụ cho khoa học máy tính của Python, hỗ trợ cho việc tính toán các mảng nhiều chiều, có kích thước lớn với các hàm đã được tối ưu áp dụng lên các mảng nhiều chiều đó. Numpy đặc biệt hữu ích khi thực hiện các hàm liên quan tới Đại Số Tuyến Tính.



NumPy

0. Khai báo thư viện

Để sử dụng Numpy thì ta phải import thư viện vào bằng lệnh như sau:

```
In [1]: import numpy as np
```

1. Creating Numpy Arrays from Python Lists

Cú pháp: np. Array ([Python list])

➔ EX:

```
In [2]: np.array ([1, 2, 3, 4])
Out[2]: array([1, 2, 3, 4])
```

Trong Python, List cho phép các phần tử trong nó khác kiểu dữ liệu, tuy nhiên với array trong Numpy, khi các kiểu dữ liệu của các phần tử khác nhau thì nó sẽ tự chuyển đổi tất cả phần tử về kiểu dữ liệu cao hơn.

➔ EX:

```
In [3]: # List trong Python: các phần tử gồm kiểu Int và Float
        [3.14, 5, 6, 7]
Out[3]: [3.14, 5, 6, 7]

In [4]: # Numpy chuyển tất cả các phần tử về kiểu Float
        np.array ([3.14, 5, 6, 7])
Out[4]: array([3.14, 5.  , 6.  , 7.  ])
```

1. Creating Numpy Arrays from Python Lists

Một điểm khác nhau nữa giữa Python List và Numpy Arrays đó chính là Numpy Array cho phép ta cụ thể hóa kiểu dữ liệu mảng ta cần.

Cú pháp: np.array ([Python list], dtype = 'kiểu dữ liệu')

➔ **EX:**

```
In [5]: # Cụ thể hóa kiểu dữ liệu mảng là kiểu Float 32 bit
        np.array ([10, 11, 12, 13], dtype = 'float32')

Out[5]: array([10., 11., 12., 13.], dtype=float32)
```

Có thể gán mảng Numpy Arrays này cho một biến bất kỳ, Python sẽ tự nhận biết kiểu dữ liệu của biến đó là Numpy Arrays.

➔ **EX:**

```
In [6]: a1 = np.array ([1, 2, 3, 4])

In [7]: type(a1)

Out[7]: numpy.ndarray
```

1. Creating Numpy Arrays from Python Lists

❑ Thuộc tính: **shape.** Thuộc tính này sẽ return cho ta biết mảng có bao nhiêu cột, bao nhiêu dòng.

➔EX:

```
In [9]: a2 = np.array([[1, 2, 3],
                       [4, 5, 6],
                       [7, 8, 9],
                       [10, 11, 12]])

In [10]: type(a2)
Out[10]: numpy.ndarray

In [11]: a2.shape
Out[11]: (4, 3)
```

1. Creating Numpy Arrays from Python Lists

❑ Thuộc tính: **ndim.** Thuộc tính này sẽ return cho ta biết số chiều của mảng

➔ **EX:**

```
In [9]: a2 = np.array([[1, 2, 3],
                       [4, 5, 6],
                       [7, 8, 9],
                       [10, 11, 12]])

In [10]: type(a2)
Out[10]: numpy.ndarray

In [11]: a2.shape
Out[11]: (4, 3)

In [12]: a2.ndim
Out[12]: 2
```

1. Creating Numpy Arrays from Python Lists

❑ Thuộc tính: **dtype.** Thuộc tính này sẽ return cho ta biết kiểu dữ liệu của các phần tử ở trong mảng.

➔EX:

```
In [9]: a2 = np.array([[1, 2, 3],
                       [4, 5, 6],
                       [7, 8, 9],
                       [10, 11, 12]])

In [10]: type(a2)
Out[10]: numpy.ndarray

In [11]: a2.shape
Out[11]: (4, 3)

In [12]: a2.ndim
Out[12]: 2

In [13]: a2.dtype
Out[13]: dtype('int32')
```

1. Creating Numpy Arrays from Python Lists

❑ Thuộc tính: **size.** Thuộc tính này sẽ return cho ta biết tổng số thành phần trong mảng.

➔EX:

In [14]:
```
a2 = np.array([[1, 2, 3],
               [4, 5, 6],
               [7, 8, 9],
               [10, 11, 12]])
a2.size
```

Out[14]: 12

2. Creating Numpy Arrays from Scratch

Tạo ra mảng Numpy Arrays từ các **Build – in Function** (hàm có sẵn) của Numpy cung cấp như: zeros, ones, full, arrange, linspace

❑ Hàm zeros: Dùng để tạo mảng với các phần tử đều là số 0

➔ **Cú pháp:** np.zeros ([m, n]) với m, n là số hàng, cột của mảng cần tạo, kiểu dữ liệu mặc định của các phần tử của hàm là kiểu float.

➔ **EX:**

```
In [15]: np.zeros ([3, 6])

Out[15]: array([[0., 0., 0., 0., 0., 0.],
                [0., 0., 0., 0., 0., 0.],
                [0., 0., 0., 0., 0., 0.]])
```

2. Creating Numpy Arrays from Scratch

Có thể thay đổi kiểu dữ liệu của các phần tử của hàm zeros:

```
In [17]: np.zeros ([3, 6], dtype = int)

Out[17]: array([[0, 0, 0, 0, 0, 0],
                [0, 0, 0, 0, 0, 0],
                [0, 0, 0, 0, 0, 0]])
```

❑ Hàm ones: Dùng để tạo mảng với các phần tử đều là số 1

➔ **Cú pháp:** np.ones ([m, n]) với m, n là số hàng, cột của mảng cần tạo, kiểu dữ liệu mặc định của các phần tử của hàm là kiểu float.

➔ **EX:**

```
In [21]: np.ones ([2, 4], dtype=float)

Out[21]: array([[1., 1., 1., 1.],
                [1., 1., 1., 1.]])
```

2. Creating Numpy Arrays from Scratch

❑ **Hàm arange:** Dùng để tạo mảng với các phần tử nằm trong một giới hạn cố định

➔ **Cú pháp:** np.arrange (start, stop, step, dtype)

➔ **EX:**

```
In [22]: np.arange (0, 20, 2)
Out[22]: array([ 0,  2,  4,  6,  8, 10, 12, 14, 16, 18])
```

❑ **Hàm full:** Dùng để tạo ra một mảng với các phần tử do người lập trình tùy chọn

➔ **Cú pháp:** np.full (shape, fill_value, dtype)

➔ **EX:**

```
In [24]: np.full((2,3), 1.7)
Out[24]: array([[1.7, 1.7, 1.7],
                [1.7, 1.7, 1.7]])
```

2. Creating Numpy Arrays from Scratch

❑ **Hàm linspace:** Dùng để tạo mảng với các phần tử được phân chia một cách tuyến tính

➔ **Cú pháp:** np.arrange (start, stop, step, dtype)

➔ **EX:**

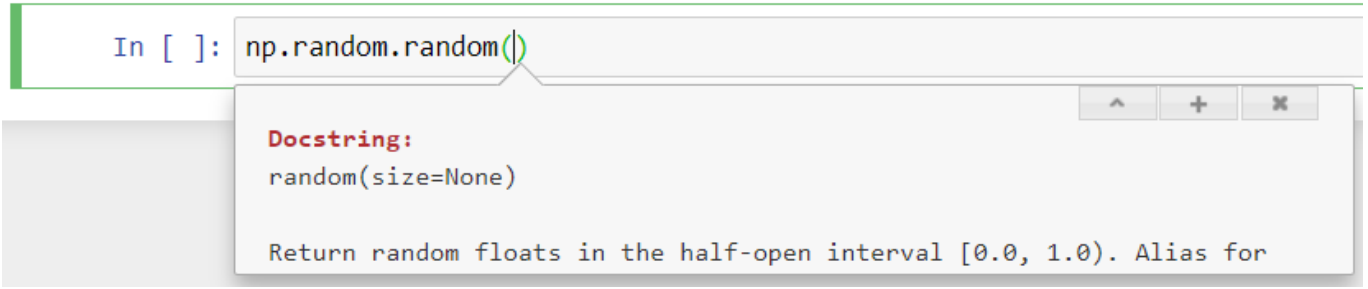
```
In [26]: np.linspace (0, 1, 5)
Out[26]: array([0.  , 0.25, 0.5 , 0.75, 1.  ])
```

Có thể hiểu ví dụ trên, step = 5, nên hàm tạo ra một mảng gồm 5 phần tử (sẽ có 4 khoảng), mỗi phần tử được chia đều nhau trong khoảng từ [0, 1], do đó, khoảng cách giữa mỗi phần tử là ¼ = 0.25.

3. Random

Hàm Random giúp ta tạo ra một mảng Numpy Array với các thành phần ngẫu nhiên tùy vào việc ta định nghĩa.

- **Option 1:** gọi chính hàm random dung để tạo ra một Numpy Array với các phần tử nằm trong khoảng từ 0 – 1.



➔ EX:

```
In [29]: np.random.random((4,4))

Out[29]: array([[0.71986746, 0.43339121, 0.47186737, 0.36701687],
                [0.85683811, 0.39815273, 0.56983277, 0.72872782],
                [0.10752187, 0.29457388, 0.82284637, 0.40592936],
                [0.65120889, 0.27143762, 0.70277975, 0.15959905]])
```

3. Random

Trong những lần gọi hàm random trên ở những thời điểm khác nhau sẽ cho ra một Numpy Arrays với các phần tử là khác nhau.

```
In [29]: np.random.random((4,4))

Out[29]: array([[0.71986746, 0.43339121, 0.47186737, 0.36701687],
                [0.85683811, 0.39815273, 0.56983277, 0.72872782],
                [0.10752187, 0.29457388, 0.82284637, 0.40592936],
                [0.65120889, 0.27143762, 0.70277975, 0.15959905]])

In [30]: np.random.random((4,4))

Out[30]: array([[0.45331703, 0.77641348, 0.45853394, 0.22015672],
                [0.42647452, 0.21758381, 0.38855397, 0.77514994],
                [0.46255265, 0.76658602, 0.45353609, 0.70868172],
                [0.17899981, 0.42282576, 0.69449495, 0.80764941]])
```

3. Random

Để giữ nguyên các giá trị phần tử được tạo ra như nhau khi thực thi trên các máy tính khác nhau, ta làm như sau:

```
In [31]: np.random.seed(0)
         np.random.random((4,4))

Out[31]: array([[0.5488135 , 0.71518937, 0.60276338, 0.54488318],
                [0.4236548 , 0.64589411, 0.43758721, 0.891773  ],
                [0.96366276, 0.38344152, 0.79172504, 0.52889492],
                [0.56804456, 0.92559664, 0.07103606, 0.0871293 ]])
```

❑ **Option 2:** np.random.normal () tạo ra một Numpy Arrays với các phần tử được xác định theo phân bố Gaussian

```
In [ ]: np.random.normal()
```

Docstring:
normal(loc=0.0, scale=1.0, size=None)

Draw random samples from a normal (Gaussian) distribution.

3. Random

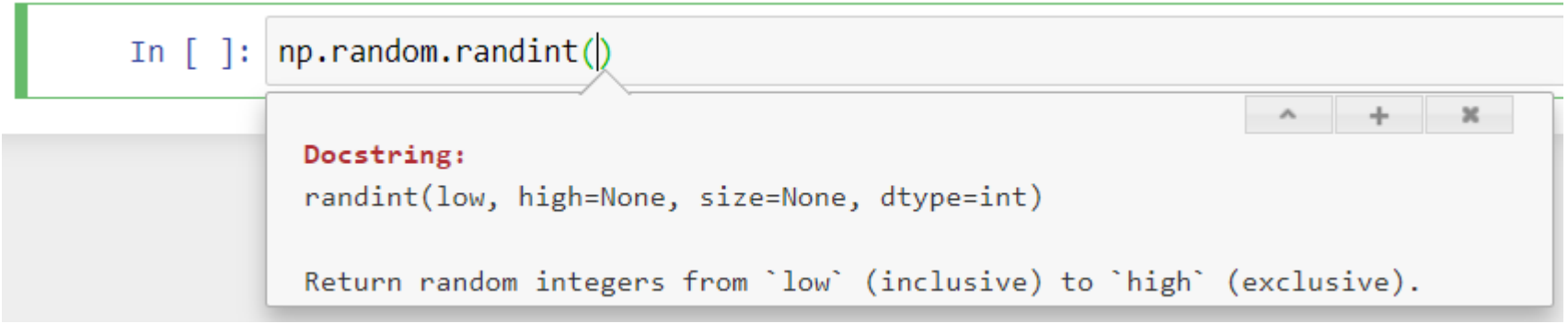
- **Option 2:** np.random.normal () tạo ra một Numpy Arrays với các phần tử được xác định theo phân bố Gaussian

Hàm có 3 tham số truyền vào: trung bình – độ lệch chuẩn – kích thước ma trận cần tạo

➔ **EX:**

```
In [36]: np.random.normal(0, 1, (3, 3))

Out[36]: array([[ 0.44386323,  0.33367433,  1.49407907],
                [-0.20515826,  0.3130677 , -0.85409574],
                [-2.55298982,  0.6536186 ,  0.8644362 ]])
```

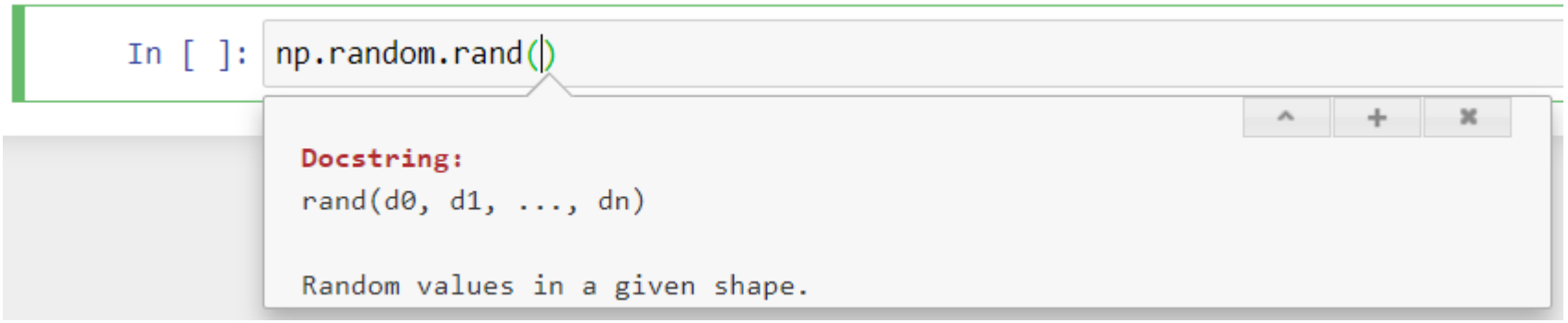
- **Option 3:** np.random.randint () tạo ra một Numpy Arrays với các phần tử là các số nguyên và được lấy ngẫu nhiên trong một khoảng xác định [LOW, HIGH). Không lấy giá trị High.



➔ EX:

```
In [2]: np.random.randint(0, 10, (4,5))
Out[2]: array([[6, 5, 6, 5, 8],
               [2, 1, 7, 9, 5],
               [0, 2, 0, 7, 4],
               [1, 8, 4, 8, 4]])
```

- **Option 4:** np.random.rand () tạo ra một Numpy Arrays với các chiều mong muốn mà không cần tạo một shape (m, n).



➔ EX:

```
In [3]: np.random.rand (4,4)
Out[3]: array([[0.07120242, 0.44472008, 0.36118384, 0.00237133],
               [0.77586153, 0.31581746, 0.19820107, 0.31858274],
               [0.26019491, 0.60518547, 0.92793534, 0.5700319 ],
               [0.05520441, 0.29008136, 0.52100686, 0.27013199]])
```

4.1. One – dimensional subarray

- Tạo 1 array có 6 thành phần, các thành phần mang kiểu dữ liệu int.

```
In [5]: x1 = np.random.randint(0, 20, size = 6)

In [6]: x1
Out[6]: array([ 4, 19,  6,  1,  4, 14])
```

- Muốn tiếp cận phần tử ở vị trí thứ 4 ở array, ta dùng index: x1 [4]

```
In [6]: x1
Out[6]: array([ 4, 19,  6,  1,  4, 14])

In [7]: x1[4]
Out[7]: 4
```

4.1. One – dimensional subarray

- Tiếp cận giá trị đầu và cuối của mảng:

```
In [6]: x1
Out[6]: array([ 4, 19,  6,  1,  4, 14])

In [7]: x1[4]
Out[7]: 4

In [16]: # return giá trị đầu trong array:
         x1[0]
Out[16]: 4

In [17]: # return giá trị cuối trong array:
         x1[-1]
Out[17]: 14

In [18]: x1 [len(x1)-1]
Out[18]: 14
```

4.2. Multi – dimensional array

- Tạo mảng array có kích thước (3, 4), mỗi phần tử mang kiểu dữ liệu int

```
In [20]: x2 = np.random.randint(20, size = (3,4))

In [21]: x2

Out[21]: array([[11,  7, 15, 18],
                [13, 17,  6,  4],
                [16, 15, 11, 16]])
```

- Giả sử muốn truy cập giá trị 6 trong mảng. 6 ở vị trí hàng 1 cột 2

```
In [21]: x2

Out[21]: array([[11,  7, 15, 18],
                [13, 17,  6,  4],
                [16, 15, 11, 16]])

In [22]: x2 [1,2]

Out[22]: 6
```

4.2. Multi – dimensional array

- Ngoài ra, ta còn có thể thay đổi một phần tử trong mảng ban đầu. Giả sử thay đổi giá trị 6 ở vị trí hàng 1 cột 2 thành 11.

```
In [21]: x2
Out[21]: array([[11,  7, 15, 18],
                [13, 17,  6,  4],
                [16, 15, 11, 16]])

In [22]: x2 [1,2]
Out[22]: 6

In [23]: x2[1, 2] = 19

In [24]: x2
Out[24]: array([[11,  7, 15, 18],
                [13, 17, 19,  4],
                [16, 15, 11, 16]])
```

4.3. Slicing

Cú pháp cắt array: x [start : stop : step]
Kết quả mảng sẽ được cắt từ vị trí có index = "start" đến vị trí có index = "stop – 1"

➔ **EX1:** Cắt 4 ký tự đầu tiên của mảng x1

```
In [25]: x1
Out[25]: array([ 4, 19,  6,  1,  4, 14])

In [26]: x1 [0:4:1]
Out[26]: array([ 4, 19,  6,  1])
```

4.3. Slicing

➔ **EX2:** Cắt 3 ký tự 6, 1, 4 của mảng x1. Ta có start = 2, stop = 5

```
In [25]: x1
Out[25]: array([ 4, 19,  6,  1,  4, 14])

In [27]: x1 [2:5:1]
Out[27]: array([6, 1, 4])
```

➔ **EX3:** Cắt các ký tự bắt đầu từ đầu array và xen kẽ nhau

```
In [25]: x1
Out[25]: array([ 4, 19,  6,  1,  4, 14])

In [28]: x1 [::2]
Out[28]: array([4, 6, 4])
```

4.3. Slicing

➔ **EX4:** Cắt lấy 2 hàng đầu và 3 cột đầu tiên của mảng x2.

```
In [29]: x2
```

```
Out[29]: array([[11,  7, 15, 18],
                [13, 17, 19,  4],
                [16, 15, 11, 16]])
```

```
In [30]: # Lấy 2 hàng đầu và 3 cột đầu tiên
         x2 [:2, :3]
```

```
Out[30]: array([[11,  7, 15],
                [13, 17, 19]])
```

4.3. Slicing

➔ **EX5:** Cắt lấy cả 2 hàng đầu và 2 cột đầu tiên của mảng x2.

```
In [31]: x2
```

```
Out[31]: array([[11,  7, 15, 18],
                [13, 17, 19,  4],
                [16, 15, 11, 16]])
```

```
In [34]: # Lấy hết hai hàng đầu tiên và 2 cột đầu tiên
         x2 [:,:2]
```

```
Out[34]: array([[11,  7],
                [13, 17],
                [16, 15]])
```

5. Reshaping of Arrays & Transpose

5.1. Reshaping: Chuyển đổi hình dạng của Numpy Array sang một hình dạng khác

- Tạo một array có 9 phần tử, có thể thấy grid là một ma trận có 9 hàng.

```
In [39]: grid = np.arange (1,10)

In [40]: grid
Out[40]: array([1, 2, 3, 4, 5, 6, 7, 8, 9])

In [41]: grid.shape
Out[41]: (9,)
```

- Cách chuyển grid về ma trận khác, ví dụ ma trận 3x3:

```
In [42]: grid.reshape((3,3))
Out[42]: array([[1, 2, 3],
                [4, 5, 6],
                [7, 8, 9]])
```

5. Reshaping of Arrays & Transpose

5.1. Reshaping: Chuyển đổi hình dạng của Numpy Array sang một hình dạng khác

- Tạo 1 colum vector có 3 hàng:

```
In [43]: x = np.array ([1, 2, 3])

In [44]: x
Out[44]: array([1, 2, 3])

In [45]: x.shape
Out[45]: (3,)
```

- Chuyển colum vector trên thành 1 vector 1 hàng, 3 cột

```
In [46]: x.reshape((1,3))
Out[46]: array([[1, 2, 3]])

In [48]: x.reshape((1,3)).shape
Out[48]: (1, 3)
```

5.2. Transpose: Chuyển đổi từ cột thành hàng và từ hàng thành cột

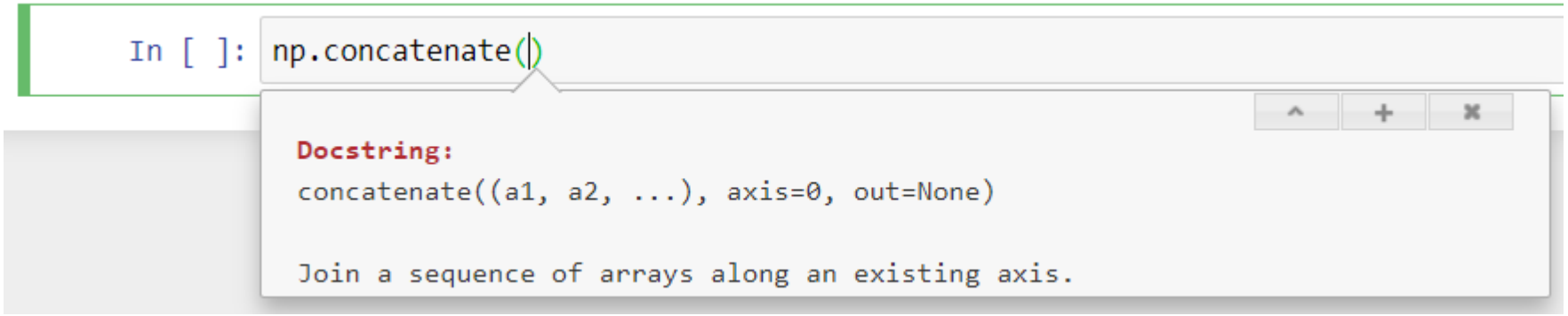
```
In [50]: x
```

```
Out[50]: array([[1, 2, 3],
                [4, 5, 6]])
```

```
In [51]: # Transpose
         x.T
```

```
Out[51]: array([[1, 4],
                [2, 5],
                [3, 6]])
```

6.1. **Concatenation**: Nối các Numpy Array lại với nhau



Trong đó:

a1, a2, ... là các Numpy Array cần gộp

axis = 0: gộp theo hàng (mặc định)

axis = 1: gộp theo cột

6.1. Concatenation

➔ EX1:

```
In [52]: x = np.array ([1, 2, 3])
         y = np.array ([3, 2, 1])

In [53]: np.concatenate((x, y))

Out[53]: array([1, 2, 3, 3, 2, 1])
```

➔ **EX2:** axis = 0 gộp theo hàng, hay cột – cột tương ứng

```
In [55]: grid = np.array ([[1, 2, 3],[4, 5, 6]])
         grid

Out[55]: array([[1, 2, 3],
                [4, 5, 6]])

In [56]: np.concatenate((grid, grid))

Out[56]: array([[1, 2, 3],
                [4, 5, 6],
                [1, 2, 3],
                [4, 5, 6]])
```

6.1. Concatenation

➔ **EX3:** axis = 1: gộp theo cột hay gộp hàng – hàng tương ứng

```
In [60]: grid = np.array ([[1, 2, 3],[4, 5, 6]])
         grid

Out[60]: array([[1, 2, 3],
                [4, 5, 6]])

In [61]: np.concatenate((grid, grid), axis = 1)

Out[61]: array([[1, 2, 3, 1, 2, 3],
                [4, 5, 6, 4, 5, 6]])
```

6.1. Concatenation

Numpy còn cung cấp cho chúng ta hàm **vstack** và **hstack** để kết nối các Array với nhau.

a) **vstack:** nối các Array theo kiểu cột - cột tương ứng với nhau

```
In [65]: x = np.array ([1, 2, 3])
         grid = np.array ([[2, 4, 6],
                           [3, 5, 7]])

In [67]: np.vstack((x, grid))

Out[67]: array([[1, 2, 3],
                [2, 4, 6],
                [3, 5, 7]])
```

6.1. Concatenation

Numpy còn cung cấp cho chúng ta hàm **vstack** và **hstack** để kết nối các Array với nhau.

a) **hstack:** nối các Array theo kiểu hàng - hàng tương ứng với nhau

```
In [72]: y = np.array ([[100],
                        [100]])
         grid = np.array ([[2, 4, 6],
                           [3, 5, 7]])

In [73]: np.hstack((y, grid))

Out[73]: array([[100,   2,   4,   6],
                [100,   3,   5,   7]])
```

6.1. Concatenation

Numpy còn cung cấp cho chúng ta hàm **vstack** và **hstack** để kết nối các Array với nhau.

a) **hstack:** nối các Array theo kiểu hàng - hàng tương ứng với nhau

```
In [72]: y = np.array ([[100],
                        [100]])
         grid = np.array ([[2, 4, 6],
                           [3, 5, 7]])

In [73]: np.hstack((y, grid))

Out[73]: array([[100,   2,   4,   6],
                [100,   3,   5,   7]])
```

Lưu ý: np.hstack ((y, grid)) # np.hstack ((grid, y))

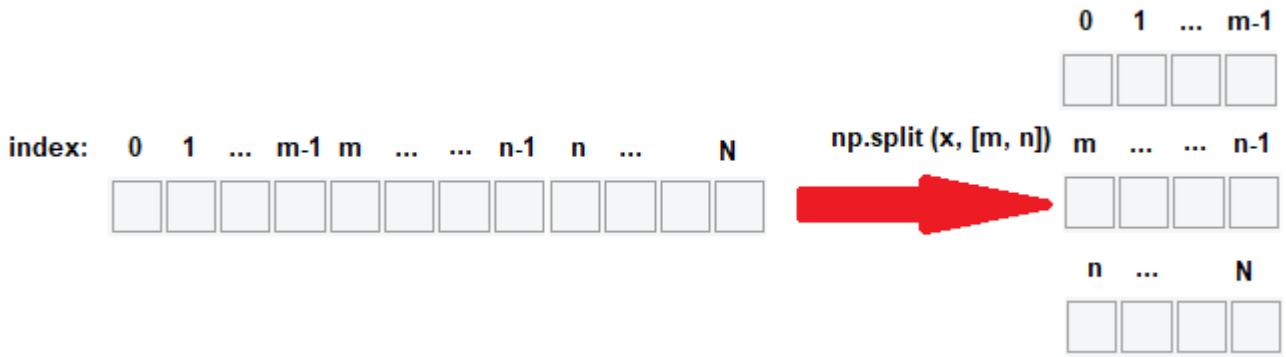
```
In [74]: np.hstack((grid, y))

Out[74]: array([[  2,   4,   6, 100],
                [  3,   5,   7, 100]])
```

6.2. **Splitting**: cắt một Numpy Array thành nhiều Array nhỏ hơn

```
In [ ]: np.split()
```

```
Signature: np.split(ary, indices_or_sections, axis=0)
Docstring:
Split an array into multiple sub-arrays as views into `ary`.
```

Cú pháp: np.split (array, [m, n])



➔ EX:

```
In [75]: x = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9])

In [76]: np.split(x, [3,5])
Out[76]: [array([1, 2, 3]), array([4, 5]), array([6, 7, 8, 9])]
```

6.2. Splitting:

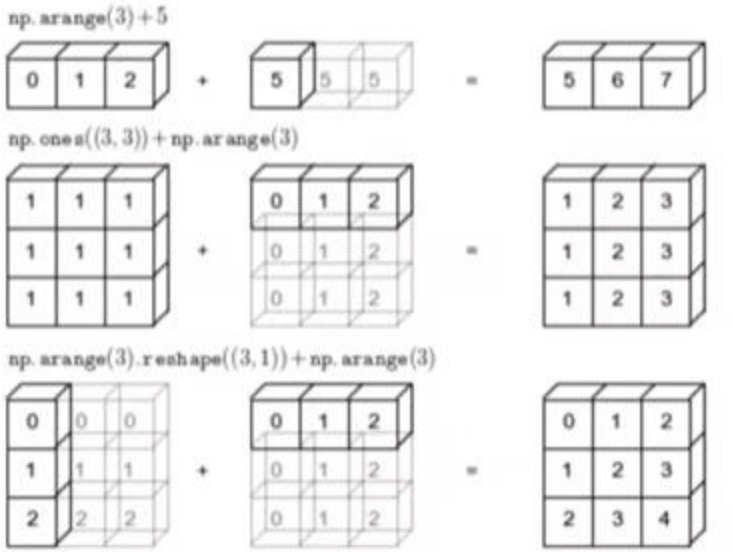
Ở ví dụ trên, có thể gán biến cho các Array nhỏ sau khi được cắt để phục vụ mục đích sử dụng khác.

```
In [75]: x = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9])

In [77]: x1, x2, x3 = np.split(x, [3,5])

In [78]: x1
Out[78]: array([1, 2, 3])

In [79]: x2
Out[79]: array([4, 5])

In [80]: x3
Out[80]: array([6, 7, 8, 9])
```

7.1. Broadcasting

Broadcasting and Vectorized operations

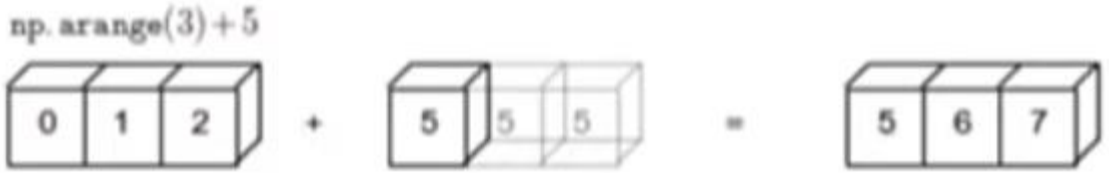
Broadcasting is simply a set of rules for applying binary ufuncs (e.g., addition, subtraction, multiplication, etc.) on arrays of different sizes.



7.1. Broadcasting

➔ EX1:

```
In [85]: a = np.arange(3)

In [86]: a
Out[86]: array([0, 1, 2])

In [87]: a+5 #Broadcasting
Out[87]: array([5, 6, 7])
```



7.1. Broadcasting

➔ EX2: Tạo thêm một ma trận b 3x3 có các phần tử là 1

```
In [89]: b = np.ones((3,3))

In [90]: b

Out[90]: array([[1., 1., 1.],
                [1., 1., 1.],
                [1., 1., 1.]])

In [91]: a.shape, b.shape

Out[91]: ((3,), (3, 3))

In [92]: a+b #Broadcasting

Out[92]: array([[1., 2., 3.],
                [1., 2., 3.],
                [1., 2., 3.]])
```

Tương tự như phép cộng, Broadcasting cũng được áp dụng cho phép nhân.

7.1. Broadcasting

➔ EX3: Tạo thêm một array c 3x1, sau đó cộng với array a 1x3

```
In [94]: c = np.arange(3).reshape((3,1))

In [95]: c

Out[95]: array([[0],
                [1],
                [2]])

In [96]: a

Out[96]: array([0, 1, 2])

In [97]: a+c

Out[97]: array([[0, 1, 2],
                [1, 2, 3],
                [2, 3, 4]])
```

8.1. np.sum ()

- Hai hàm tính tổng các phần tử trong một array của Numpy và Python

```
In [2]: list_number = [1, 2, 3]

In [3]: ll = np.array(list_number)

In [4]: ll
Out[4]: array([1, 2, 3])

In [5]: sum(ll) # Python sum ()
Out[5]: 6

In [6]: np.sum(ll) # Numpy sum ()
Out[6]: 6
```

8.1. np.sum ()

- So sánh thời gian thực thi lệnh của hai hàm tính tổng:
  + Tạo một array với rất nhiều phần tử :

```
In [7]: # Create a massive numpy array
        massive_array = np.random.random(10000)

In [8]: massive_array [:5]
Out[8]: array([0.33460855, 0.25320716, 0.07778393, 0.15988573, 0.37933025])

In [9]: massive_array.shape
Out[9]: (10000,)
```

8.1. np.sum ()

+ Để so sánh hai hàm này hàm nào tính toán nhanh hơn, Jupyter notebook hỗ trợ hàm **%timeit** + **operation**

```
In [10]: %timeit sum(massive_array) # Python buil-in function sum()
         %timeit np.sum(massive_array) # Numpy's np.sum()

         1.7 ms ± 45 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)
         8.15 µs ± 320 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)
```

+ Có thể thấy rằng, thời gian tính toán của hàm Sum do Python cung cấp là 1,7ms, còn của hàm sum trong thư viện Numpy là 8.15us thời gian tính toán nhanh hơn rất nhiều so với hàm Sum do Python cung cấp.

➔ Đó là lý do tại sao ta phải sử dụng thư viện Numpy để làm việc với mảng một chiều hay đa chiều trong quá trình học tập, nghiên cứu Machine Learning và Deep Learning.

8.2. np.mean (): Tính giá trị trung bình của các phần tử trong một array

```
In [11]: np.mean(np.array([1, 2, 3]))
Out[11]: 2.0

In [12]: np.mean(massive_array)
Out[12]: 0.49624386076199956
```

8.3. np.max() (hoặc np.min ()): Tìm ra giá trị phần tử lớn nhất (hoặc nhỏ nhất) trong các phần tử của một array

```
In [13]: np.max(np.array([1, 100, 999, 8888]))
Out[13]: 8888

In [14]: np.max(massive_array)
Out[14]: 0.9999846590750932

In [15]: np.min(np.array([1, 100, 999, 8888]))
Out[15]: 1

In [16]: np.min(massive_array)
Out[16]: 5.289690317211715e-05
```

8.4. Standard Deviation and Variance

8.4.1. Standard Deviation (độ lệch chuẩn)

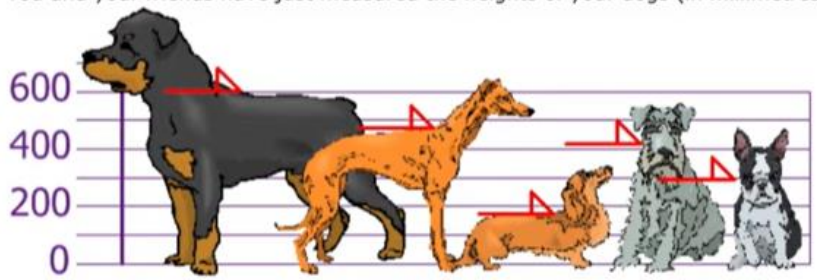
- Thước đo độ phân tán so với giá trị trung bình (mean) của dữ liệu
- Là căn bậc hai của phương sai. (square root of the variance)

8.4.2. Variance (phương sai)

- Lấy từng point dữ liệu trừ do giá trị trung bình, rồi bình phương lên và tính trung bình những giá trị đó với nhau.

Example

You and your friends have just measured the heights of your dogs (in millimetres):



The heights (at the shoulders) are: 600mm, 470mm, 170mm, 430mm and 300mm.

Find out the Mean, the Variance, and the Standard Deviation.

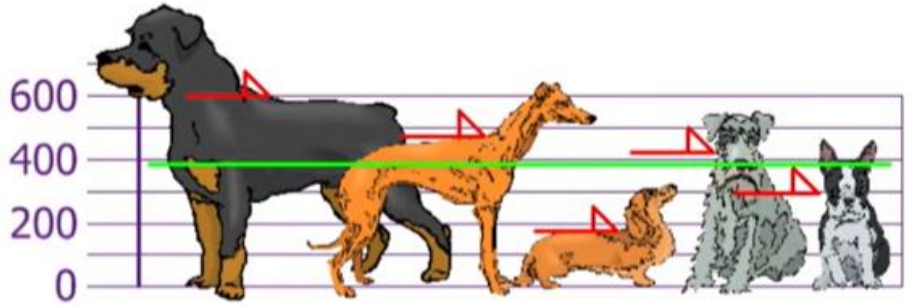
8.4. Standard Deviation and Variance

Your first step is to find the Mean:

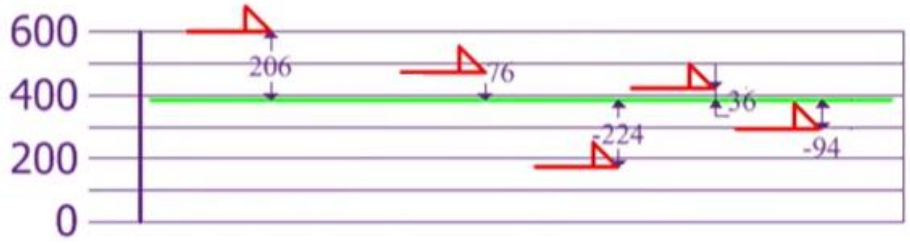
Answer:

$$\text{Mean} = \frac{600 + 470 + 170 + 430 + 300}{5}$$
$$= \frac{1970}{5}$$
$$= 394$$

so the mean (average) height is 394 mm. Let's plot this on the chart:



Now we calculate each dog's difference from the Mean:



To calculate the Variance, take each difference, square it, and then average the result:

Variance

$$\sigma^2 = \frac{206^2 + 76^2 + (-224)^2 + 36^2 + (-94)^2}{5}$$
$$= \frac{42436 + 5776 + 50176 + 1296 + 8836}{5}$$
$$= \frac{108520}{5}$$
$$= 21704$$

So the Variance is **21,704**

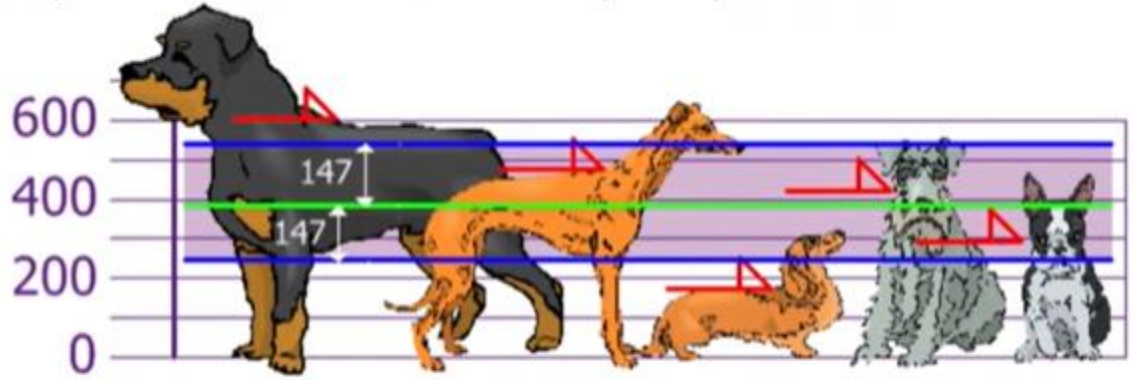
And the Standard Deviation is just the square root of Variance, so:

Standard Deviation

$$\sigma = \sqrt{21704}$$
$$= 147.32...$$
$$= \mathbf{147}\ \textit{(to the nearest mm)}$$

8.4. Standard Deviation and Variance

And the good thing about the Standard Deviation is that it is useful. Now we can show which heights are within one Standard Deviation (147mm) of the Mean:



So, using the Standard Deviation we have a "standard" way of knowing what is normal, and what is extra large or extra small.

Rottweilers **are** tall dogs. And Dachshunds **are** a bit short, right?

8.4. Standard Deviation and Variance

- Dùng Numpy để tính độ lệch chuẩn và phương sai ở ví dụ bên trên về chiều cao của những chú chó.

In [18]:
```
dog_height = [600, 470, 170, 430, 300]
dog_height = np.array(dog_height)

# Standard Deviation (độ lệch chuẩn)
dog_height_std = np.std(dog_height)
print("standard deviation: ",dog_height_std)

#Variance (phương sai)
dog_height_variance = np.var(dog_height)
print("variance: ", dog_height_variance)

# Ngoài ra, độ lệch chuẩn còn được tính bằng căn bậc hai của phương sai
dog_height_std_1 = np.sqrt(dog_height_variance)
print("standard deviation using sqrt(var): ",dog_height_std_1)
```

```
standard deviation:  147.32277488562318
variance:  21704.0
standard deviation using sqrt(var):  147.32277488562318
```

9. Sorting Arrays

Sorting Arrays là sắp xếp các thành phần trong một Arrays theo thứ tự tang dần hoặc giảm dần. Numpy cung cấp cho ta hàm **np.sort ()** dựa trên thuật toán **quicksort**

- **np.sort ():** (mặc định) sẽ sắp xếp các phần tử của Arrays theo thứ tự tăng dần

```
In [3]: x = np.array([1,2,33, 45, 98, 26, 47])
        np.sort(x)

Out[3]: array([ 1,  2, 26, 33, 45, 47, 98])
```

- **np.argsort ():** trả về một array chứa vị trí index của các phần tử đã được sắp xếp bởi hàm np.sort ()

```
In [3]: x = np.array([1,2,33, 45, 98, 26, 47])
        np.sort(x)

Out[3]: array([ 1,  2, 26, 33, 45, 47, 98])

In [6]: # A related function is argsort, which instead returns the indices of the sorted element
        np.argsort(x)

Out[6]: array([0, 1, 5, 2, 3, 6, 4], dtype=int64)
```

9. Sorting Arrays

Numpy không chỉ hỗ trợ việc sắp xếp các phần tử của mảng một chiều mà còn hỗ trợ việc sắp xếp các phần tử cho mảng đa chiều, có thể sort theo row hoặc column

❑ Sắp xếp theo chiều dọc (column): axis = 0

```
In [14]: np.random.seed(42)
         MatA = np.random.randint(0, 10, (4,6))

In [15]: MatA

Out[15]: array([[6, 3, 7, 4, 6, 9],
                [2, 6, 7, 4, 3, 7],
                [7, 2, 5, 4, 1, 7],
                [5, 1, 4, 0, 9, 5]])

In [16]: np.sort(MatA, axis = 0)

Out[16]: array([[2, 1, 4, 0, 1, 5],
                [5, 2, 5, 4, 3, 7],
                [6, 3, 7, 4, 6, 7],
                [7, 6, 7, 4, 9, 9]])
```

Numpy không chỉ hỗ trợ việc sắp xếp các phần tử của mảng một chiều mà còn hỗ trợ việc sắp xếp các phần tử cho mảng đa chiều, có thể sort theo row hoặc column

❑ Sắp xếp theo chiều ngang (row): axis = 1

```
In [14]: np.random.seed(42)
         MatA = np.random.randint(0, 10, (4,6))

In [15]: MatA

Out[15]: array([[6, 3, 7, 4, 6, 9],
                [2, 6, 7, 4, 3, 7],
                [7, 2, 5, 4, 1, 7],
                [5, 1, 4, 0, 9, 5]])

In [18]: np.sort(MatA, axis = 1)

Out[18]: array([[3, 4, 6, 6, 7, 9],
                [2, 3, 4, 6, 7, 7],
                [1, 2, 4, 5, 7, 7],
                [0, 1, 4, 5, 5, 9]])
```

Đại số tuyến tính ta thường làm việc với ma trận và nhân ma trận với nhau. Trong Numpy cũng cung cấp cho ta các hàm Build – in để thực hiện việc nhân ma trận một cách dễ dàng hơn.

❑ Tích vô hướng

In [19]:
```
A = np.array([[1, 2, 3],
              [4, 5, 6],
              [7, 8, 9]])
```

In [20]:
```
B = np.array([[6,5],
              [4,3],
              [2,1]])
```

In [21]:
```
# A(3x3) dot product B (3x2)
A.dot(B)
```

Out[21]:
```
array([[20, 14],
       [56, 41],
       [92, 68]])
```

❑ Tích vô hướng

Cũng có thể dung dấu @ để tính tích vô hướng: A@B, kết quả trả về cũng như phép tính A.dot(B)

```
In [22]: A @ B

Out[22]: array([[20, 14],
                [56, 41],
                [92, 68]])
```

B (3x2) không thể nhân vô hướng với A(3x3) vì không cùng dimension. Nhưng khi chuyển vị B (transpose) thì B chuyển vị có thể nhân vô hướng với A.

```
In [26]: (B.T)@A

Out[26]: array([[36, 48, 60],
                [24, 33, 42]])
```

❑ Dot Product Example

In [27]:
```
# Number of jars sold
np.random.seed(0)

sales_amounts = np.random.randint(20, size=(5,3))
```

In [28]:
```
sales_amounts
```

Out[28]:
```
array([[12, 15,  0],
       [ 3,  3,  7],
       [ 9, 19, 18],
       [ 4,  6, 12],
       [ 1,  6,  7]])
```

In [29]:
```
# Creat weekly_sales DataFrame
import pandas as pd

weekly_sales = pd.DataFrame(sales_amounts, index = ["Mon", "Tues", "Wed", "Thurs", "Fri"],
                                           columns = ["Almond Butter", "Peanut Butter", "Cashew Butter"])
```

In [30]:
```
weekly_sales
```

Out[30]:

	Almond Butter	Peanut Butter	Cashew Butter
Mon	12	15	0
Tues	3	3	7
Wed	9	19	18
Thurs	4	6	12
Fri	1	6	7

❑ Dot Product Example

In [31]:
```
# Create a price array
prices = np.array([10, 8, 12])
```

In [32]:
```
ter_prices = pd.DataFrame(prices.reshape(1,3), index = ["Prices"], columns = ["Almond Butter", "Peanut Butter", "Cashew Butter"])
```

In [33]:
```
butter_prices
```

Out[33]:

	Almond Butter	Peanut Butter	Cashew Butter
Prices	10	8	12

In [34]:
```
weekly_sales.shape, butter_prices.shape
```

Out[34]:
```
((5, 3), (1, 3))
```

In [35]:
```
total_prices = weekly_sales @ butter_prices.T
```

In [36]:
```
total_prices
```

Out[36]:

	Prices
Mon	240
Tues	138
Wed	458
Thurs	232
Fri	142

❑ Dot Product Example

In [37]:
```
weekly_sales["Total Prices"] =total_prices
```

In [38]:
```
weekly_sales
```

Out[38]:

	Almond Butter	Peanut Butter	Cashew Butter	Total Prices
Mon	12	15	0	240
Tues	3	3	7	138
Wed	9	19	18	458
Thurs	4	6	12	232
Fri	1	6	7	142

THE END