Pytorch簡介

黃威澄

Outline

- Data structure
  - List
  - Dictionary
  - Tuple
- Pytorch

Data structure

- List
- Dictionary
- Tuple

Lists(1)

- A list is the Python equivalent of an array
- resizeable and contain elements of different types.

```
xs = [3, 1, 2]    # Create a list
print(xs, xs[2])
print(xs[-1])      # Negative indices count from the end of the list; prints "2"
```

```
[3, 1, 2] 2 2
```

```
xs[2] = 'foo'     # Lists can contain elements of different types
print(xs)
```

```
[3, 1, 'foo']
```

```
xs.append('bar') # Add a new element to the end of the list
print(xs)
```

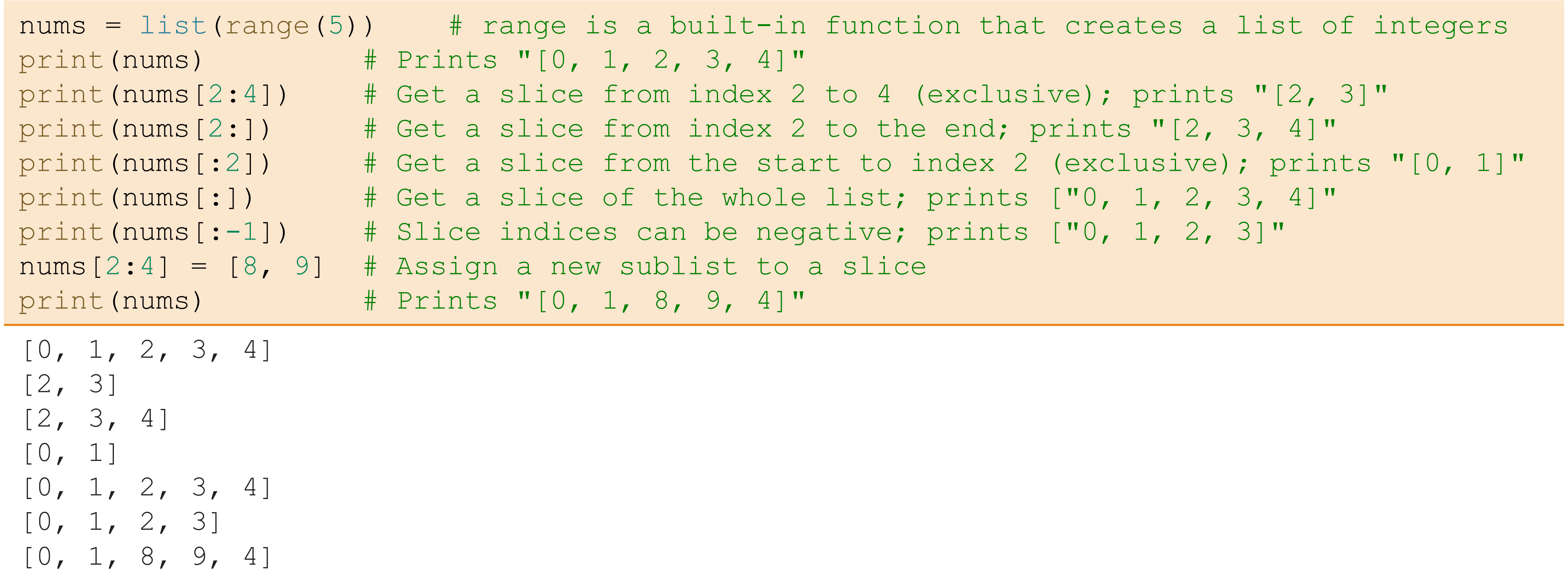
```
[3, 1, 'foo', 'bar']
```

```
x = xs.pop()      # Remove and return the last element of the list
print(x, xs)
```

```
bar [3, 1, 'foo']
```

List(2)-Slicing

- Examples:



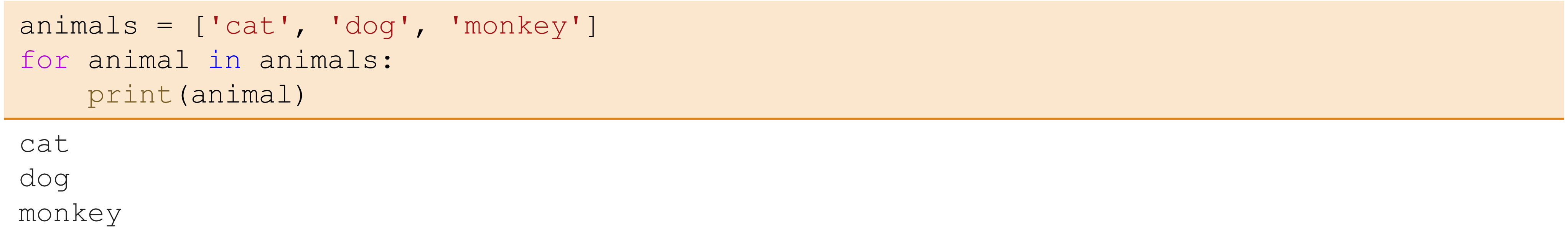
```
nums = list(range(5))    # range is a built-in function that creates a list of integers
print(nums)         # Prints "[0, 1, 2, 3, 4]"
print(nums[2:4])    # Get a slice from index 2 to 4 (exclusive); prints "[2, 3]"
print(nums[2:])     # Get a slice from index 2 to the end; prints "[2, 3, 4]"
print(nums[:2])     # Get a slice from the start to index 2 (exclusive); prints "[0, 1]"
print(nums[:])      # Get a slice of the whole list; prints ["0, 1, 2, 3, 4]"
print(nums[:-1])    # Slice indices can be negative; prints ["0, 1, 2, 3]"
nums[2:4] = [8, 9]  # Assign a new sublist to a slice
print(nums)         # Prints "[0, 1, 8, 9, 4]"
```

```
[0, 1, 2, 3, 4]
[2, 3]
[2, 3, 4]
[0, 1]
[0, 1, 2, 3, 4]
[0, 1, 2, 3]
[0, 1, 8, 9, 4]
```

List(3)-Loop

```
for x in sequence:
# sequence: python list, range(), enumerate…
  # 放要執行的東西(記得縮排)
```

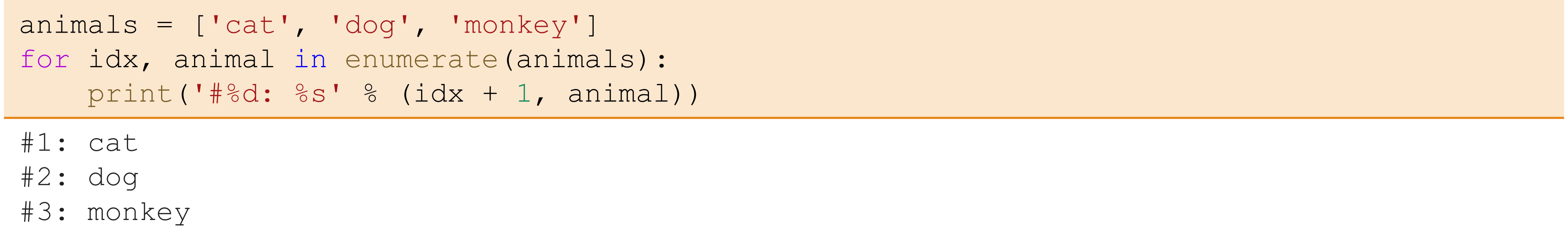
- Examples:



```
animals = ['cat', 'dog', 'monkey']
for animal in animals:
    print(animal)
```

```
cat
dog
monkey
```

- enumerate function.



```
animals = ['cat', 'dog', 'monkey']
for idx, animal in enumerate(animals):
    print('#%d: %s' % (idx + 1, animal))
```

```
#1: cat
#2: dog
#3: monkey
```

List comprehensions

- Examples:

```
x = []
for i in range(5):
    x.append(i)
print(x)
```

```
[0, 1, 2, 3, 4]
```

- List comprehensions can also contain conditions:

```
x = []
for i in range(5):
  if i%2 == 0:
    x.append(i)
print(x)
```

```
[0, 2, 4]
```

Dictionaries

- A dictionary stores (key, value) pairs.
- Examples:

```
d = {'cat': 'cute', 'dog': 'furry'}  # Create a new dictionary with some data
print(d['cat'])       # Get an entry from a dictionary; prints "cute"
print('cat' in d)     # Check if a dictionary has a given key; prints "True"
```

```
cute
True
```

```
d['fish'] = 'wet'    # Set an entry in a dictionary
print(d['fish'])      # Prints "wet"
```

```
wet
```

```
print(d.get('fish', 'N/A'))    # Get an element with a default; prints "wet"
del d['fish']        # Remove an element from a dictionary
print(d.get('fish', 'N/A')) # "fish" is no longer a key; prints "N/A"
```

```
wet
N/A
```

Dictionaries comprehensions

- Example:

```
x = {i: 2*i for i in range(i) if i%2 == 0}
print(x)
```

```
{0: 0, 2: 4}
```

Tuple(1)

- Same as lists, but tuples are immutable.
- Tuple, ex: (‘a’, ‘b’)
- List, ex: [‘a’, ‘b’]
- Example:

```
score = "A+"          # type: string
print(type(score))
score = tuple(score) # type: tuple
print(score)
print(type(score))
score = "A+", # a tuple with a single element must have a comma
print(score)
```

```
<class 'str’>
('A', '+’)
<class 'tuple’>
('A+',)
```

Tuple(2)

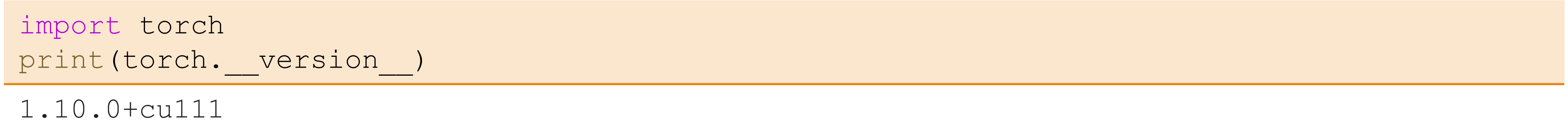
- Example(cont’d):

```
score = ("A+","A","A-") # enclosed in parentheses
print(score)
print(type(score))
num = {"A+":95, "A":85, "A-":80}
print("A is corresponding to",num[score[1]]) # tuples can be keys  of dictionaries
score[1] = "A-"
```

```
('A+', 'A', 'A-’)
<class 'tuple’>
A is corresponding to 85
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-24-0c2ca30bf646> in <module>()
     13 print("A is corresponding to",num[score[1]])
     14
---> 15 score[1] = "A-"
TypeError: 'tuple' object does not support item assignment
```

Pytorch

- PyTorch 開源機器學習架構.
  - 多維 Tensor 物件
  - GPU加速
  - 自動微分引擎,計算出導數
  - 乾淨的應用程式開發介面
- Import Pytorch



```
import torch
print(torch.__version__)
```

```
1.10.0+cu111
```

Creating and Accessing tensors(1)

- Example:

```
# Create a rank 1 tensor from a Python list
a = torch.tensor([1, 2, 3])
print(a)
print('type(a): ', type(a))
print('rank of a: ', a.dim())
print('a.shape: ', a.shape)
print('a[0]: ', a[0])
a[1] = 10
print(a)
```

```
tensor([1, 2, 3])
type(a): <class 'torch.Tensor’>
rank of a: 1
a.shape: torch.Size([3])
tensor(1)

tensor([ 1, 10, 3])
```

```
# Create a two-dimensional tensor
b = torch.tensor([[1, 2, 3], [4, 5, 5]])
print(b)

print('rank of b:', b.dim())
print('b.shape: ', b.shape)
print('b[0, 1]:', b[0, 1])
b[1, 1] = 100
print(b)
```

```
tensor([[1, 2, 3],
        [4, 5, 5]])
rank of b: 2
b.shape: torch.Size([2, 3])
b[0, 1]: tensor(2)

tensor([[ 1, 2, 3],
        [ 4, 100, 5]])
```

Creating and Accessing tensors(2)

- Example:

```
# Create a tensor of all zeros
a = torch.zeros(2, 3)
print('tensor of zeros:')
print(a)
# Create a tensor of all ones
b = torch.ones(1, 2)
print('\ntensor of ones:')
print(b)
# Create a 3x3 identity matrix
c = torch.eye(3)
print('\nidentity matrix:')
print(c)

# Tensor of random values
d = torch.rand(2, 3)
print('\nrandom tensor:')
print(d)
```

```
tensor of zeros:
tensor([[0., 0., 0.],
        [0., 0., 0.]])

tensor of ones:
tensor([[1., 1.]])


identity matrix:
tensor([[1., 0., 0.],
        [0., 1., 0.],
        [0., 0., 1.]])

random tensor:
tensor([[0.1874, 0.0322, 0.9912],
        [0.7445, 0.3450, 0.1959]])
```

Tensor indexing-Slice indexing

- Example

```
# Create the following rank 2 tensor with shape (3, 4)
a = torch.tensor([[1,2,3,4], [5,6,7,8], [9,10,11,12]])

row_r1 = a[1, :]# Rank 1 view of the second row of a
row_r2 = a[1:2, :]  # Rank 2 view of the second row of a
print(row_r1, row_r1.shape)
print(row_r2, row_r2.shape)

# We can make the same distinction when accessing column
s:
col_r1 = a[:, 1]
col_r2 = a[:, 1:2]
print(col_r1, col_r1.shape)
print(col_r2, col_r2.shape)
```

```
tensor([5, 6, 7, 8])torch.Size([4])
tensor([[5, 6, 7, 8]]) torch.Size([1, 4])
```

```
tensor([ 2, 6, 10]) torch.Size([3])
tensor([[ 2],
        [ 6],
        [10]]) torch.Size([3, 1])
```

Tensor indexing-Integer tensor indexing

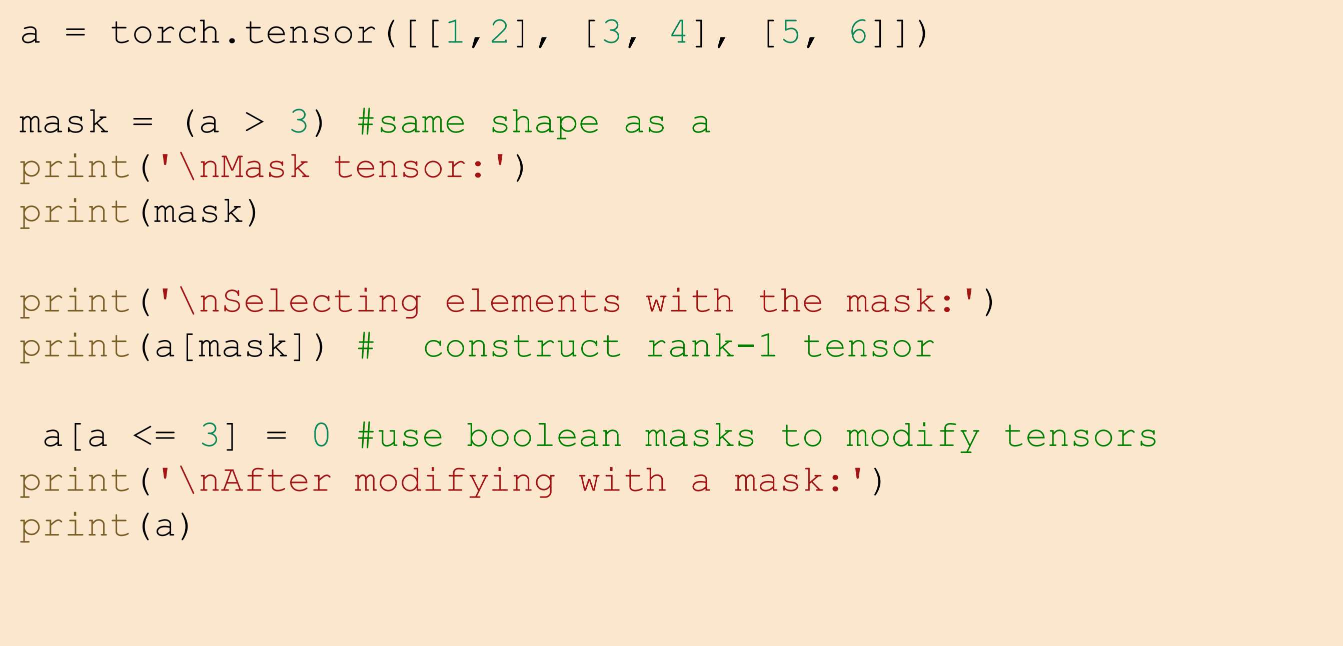
- Example

```
a = torch.tensor([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11,
12]])

# Create a new tensor of shape (5, 4) by reordering rows
idx = [0, 0, 2, 1, 1]
          # index arrays can be Python lists of integers
print(a[idx])

# Create a new tensor of shape (3, 4) by reversing the
#columns from a
idx = torch.tensor([3, 2, 1, 0])
             # Index arrays can be int64 torch tensors
print(a[:, idx])
```

```
tensor([[ 1,  2,  3,  4],
        [ 1,  2,  3,  4],
        [ 9, 10, 11, 12],
        [ 5,  6,  7,  8],
        [ 5,  6,  7,  8]])

tensor([[ 4,  3,  2,  1],
        [ 8,  7,  6,  5],
        [12, 11, 10,  9]])
```

Tensor indexing-Boolean tensor indexing

- Example



```
a = torch.tensor([[1,2], [3, 4], [5, 6]])

mask = (a > 3) #same shape as a
print('\nMask tensor:')
print(mask)

print('\nSelecting elements with the mask:')
print(a[mask]) #  construct rank-1 tensor

 a[a <= 3] = 0 #use boolean masks to modify tensors
print('\nAfter modifying with a mask:')
print(a)
```

```
Mask tensor:
tensor([[False, False],
        [False,  True],
        [ True,  True]])
Selecting elements with the mask:
tensor([4, 5, 6])

After modifying with a mask:
tensor([[0, 0],
        [0, 4],
        [5, 6]])
```

Reshaping operations(1)-view

- Example

```
import torch
x0 = torch.tensor([[1, 2, 3, 4], [5, 6, 7, 8]])

x1 = x0.view(2,2,2)
print(x1)

x2 = x0.view(-1,2,2) #same as x1
print(x2)
```

```
tensor([[[1, 2],
         [3, 4]],

        [[5, 6],
         [7, 8]]])
```

Reshaping operations(2)-transpose

- Example

```
x = torch.tensor([[1, 2, 3], [4, 5, 6]])
print(torch.t(x))
print(x.t())
```

```
tensor([[1, 4],
        [2, 5],
        [3, 6]])
```

```
# Create a tensor of shape (2, 3, 4)
x0 = torch.tensor([
     [[1,  2,  3,  4],
      [5,  6,  7,  8],
      [9, 10, 11, 12]],
     [[13, 14, 15, 16],
      [17, 18, 19, 20],
      [21, 22, 23, 24]]])

# Swap axes 1 and 2; shape is (2, 4, 3)
x1 = x0.transpose(1, 2)
print(x1)
```

```
tensor([[[ 1, 5, 9],
         [ 2, 6, 10],
         [ 3, 7, 11],
         [ 4, 8, 12]],
        [[13, 17, 21],
         [14, 18, 22],
         [15, 19, 23],
         [16, 20, 24]]])
```

Reshaping operations(3)-permute

- Example

```
# Create a tensor of shape (2, 3, 4)
x0 = torch.tensor([
     [[1,  2,  3,  4],
      [5,  6,  7,  8],
      [9, 10, 11, 12]],
     [[13, 14, 15, 16],
      [17, 18, 19, 20],
      [21, 22, 23, 24]]])

# Permute axes; the argument (1, 2, 0) means:
# - Make the old dimension 1 appear at dimension 0;
# - Make the old dimension 2 appear at dimension 1;
# - Make the old dimension 0 appear at dimension 2
# This results in a tensor of shape (3, 4, 2)
x2 = x0.permute(1, 2, 0)
print(x2)
```

```
tensor([[[ 1, 13],
         [ 2, 14],
         [ 3, 15],
         [ 4, 16]],
        [[ 5, 17],
         [ 6, 18],
         [ 7, 19],
         [ 8, 20]],
        [[ 9, 21],
         [10, 22],
         [11, 23],
         [12, 24]]])
```

Tensor operations-Elementwise operations(1)

- Example:

```
x = torch.tensor([[1, 2, 3, 4]], dtype=torch.float32)
y = torch.tensor([[5, 6, 7, 8]], dtype=torch.float32)

# Elementwise sum; all give the same result
print(x + y)
print(torch.add(x, y))
print(x.add(y))

# Elementwise difference
print(x - y)
print(torch.sub(x, y))
print(x.sub(y))
```

```
tensor([[ 6., 8., 10., 12.]])
```

```
tensor([[-4., -4., -4., -4.]])
```

Tensor operations-Elementwise operations(2)

- Example: * 是逐元素執行, 不是矩陣乘法.

```
x = torch.tensor([[1, 2, 3, 4]], dtype=torch.float32)
y = torch.tensor([[5, 6, 7, 8]], dtype=torch.float32)

# Elementwise product
print(x * y)
print(torch.mul(x, y))
print(x.mul(y))

# Elementwise division
print(x / y)
print(torch.div(x, y))
print(x.div(y))
```

```
tensor([[ 5., 12., 21., 32.]])
```

```
tensor([[0.2000, 0.3333, 0.4286, 0.5000]])
```

Tensor operations-Elementwise operations(3)

- Example:

```
x = torch.tensor([1, 2], dtype=torch.float32)
y = torch.tensor([3, 4], dtype=torch.float32)

# Computes the dot product of two 1D tensors.
# must be vector
print(torch.dot(x,y))

z=torch.tensor([[1, 2], [4, 5]])
w = torch.tensor([[6, 7], [8, 9]])

# performs a matrix multiplication without broadcasting
print(torch.mm(z,w))

#Performs a matrix-vector product
print(torch.mv(z,x))
```

```
tensor(11)

tensor([[22, 25],
        [64, 73]])

tensor([ 5, 14])
```

torch.matmul()：Matrix product of two tensors.

https://pytorch.org/docs/stable/generated/torch.matmul.html

Tensor operations-Elementwise operations(4)

- Example:

```
x = torch.tensor([[1, 2, 3, 4]], dtype=torch.float32)
y = torch.tensor([[5, 6, 7, 8]], dtype=torch.float32)

# Elementwise power
print(x ** y)
print(torch.pow(x, y))
print(x.pow(y))

# Square root
print(torch.sqrt(x))
print(x.sqrt())

# Trig functions
print(torch.sin(x))
print(x.sin())
```

```
tensor([[1.0000e+00, 6.4000e+01, 2.1870e+03,
6.5536e+04]])

tensor([[1.0000, 1.4142, 1.7321, 2.0000]])

tensor([[ 0.8415, 0.9093, 0.1411, -0.7568]])
```

Tensor operations-Matrix operations

```
x = torch.tensor([[1, 2, 3, 4]], dtype=torch.float32)
y = torch.tensor([[5, 6, 7, 8]], dtype=torch.float32)

# Elementwise power
print(x ** y)
print(torch.pow(x, y))
print(x.pow(y))

# Square root
print(torch.sqrt(x))
print(x.sqrt())

# Trig functions
print(torch.sin(x))
print(x.sin())
```

```
tensor([[1.0000e+00, 6.4000e+01, 2.1870e+03,
6.5536e+04]])
```

```
tensor([[1.0000, 1.4142, 1.7321, 2.0000]])
```

```
tensor([[ 0.8415, 0.9093, 0.1411, -0.7568]])
```

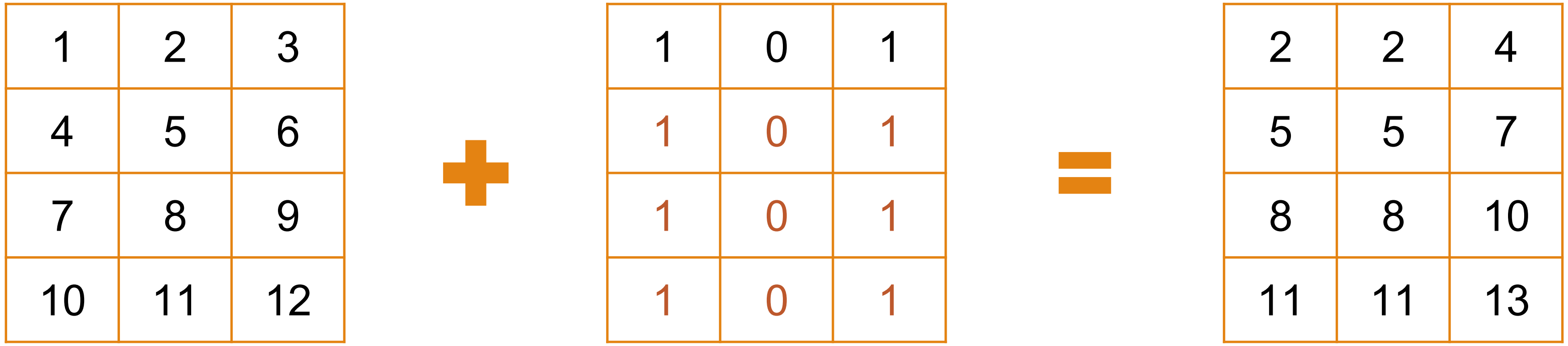
Broadcasting(1)

- Example:

```
x = torch.tensor([[1,2,3], [4,5,6], [7,8,9], [10, 11, 12]])
v = torch.tensor([1, 0, 1])
y = x + v  # Add v to each row of x using broadcasting
print(y)
```

```
tensor([[ 2,  2,  4],
        [ 5,  5,  7],
        [ 8,  8, 10],
        [11, 11, 13]])
```

- Explanation:



1	2	3
4	5	6
7	8	9
10	11	12

+

1	0	1
1	0	1
1	0	1
1	0	1

=

2	2	4
5	5	7
8	8	10
11	11	13

Broadcasting(2)

- Rules:
  - 若tensor大小不相同，在矩陣維度前加上一直到相等
  - 若矩陣維度相同，或是大小為1，兩個tensor是兼容的
  - Tensor若是兼容的可以被broadcast
  - Broadcast後大小為兩矩陣中元素最大值的大小

Broadcasting(3)

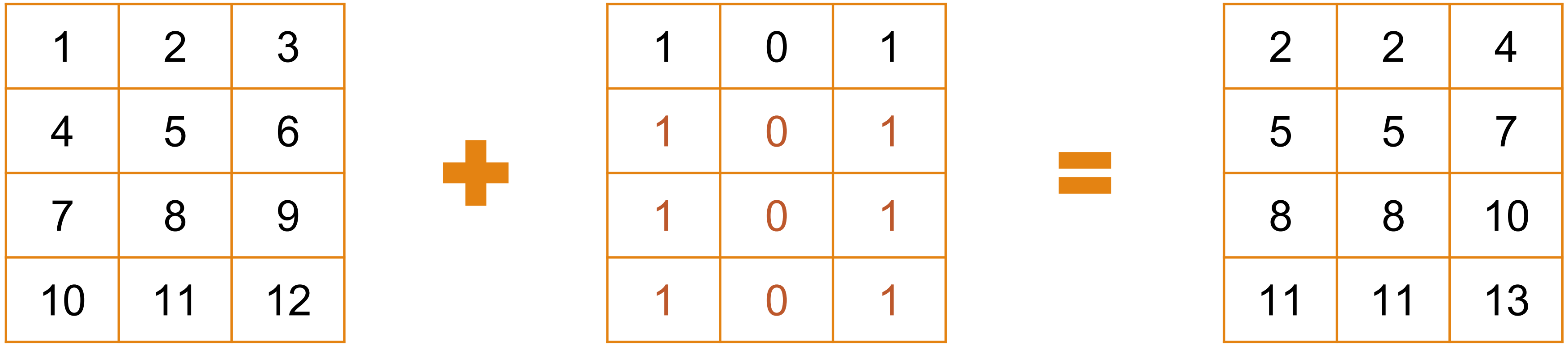
- Example:

```
x = torch.tensor([[1,2,3], [4,5,6], [7,8,9], [10, 11, 12]])
v = torch.tensor([1, 0, 1])
y = x + v  # Add v to each row of x using broadcasting
print(y)
```

```
tensor([[ 2,  2,  4],
        [ 5,  5,  7],
        [ 8,  8, 10],
        [11, 11, 13]])
```

- Explanation:

(3,) ⇨ (1,3) ⇨ (4,3)



1	2	3
4	5	6
7	8	9
10	11	12

+

1	0	1
1	0	1
1	0	1
1	0	1

=

2	2	4
5	5	7
8	8	10
11	11	13

Broadcasting(4)

- Example:

```
A = torch.zeros(2,5,3,4)
B = torch.zeros(3,1)    # (3,1) --> (1,1,3,1) --> (2,5,3,4)
print((A+B).shape)                                            torch.Size([2, 5, 3, 4])

A = torch.zeros(2,5,3,4)
B = torch.zeros(2,1,1,4) # (2,1,1,4) --> (2,5,3,4)
print((A+B).shape)                                            torch.Size([2, 5, 3, 4])

A = torch.zeros(1)  # (1,) --> (1,1) --> (3,4)
B = torch.zeros(3,4)
print((A+B).shape)                                            torch.Size([3, 4])

A = torch.zeros(2,5,3,4)
B = torch.zeros(2,4,1,4)
print((A+B).shape)      # get error!
```

Broadcasting(5)

$$A_{3\times1} \otimes B_{1\times2} = AB = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} \begin{bmatrix} b_1 & b_2 \end{bmatrix} = \begin{bmatrix} a_1 b_1 & a_1 b_2 \\ a_2 b_1 & a_2 b_2 \\ a_3 b_1 & a_3 b_2 \end{bmatrix}$$

- Example:

```
# Compute outer product of vectors
v = torch.tensor([1, 2, 3])  # v has shape (3,)
w = torch.tensor([4, 5])     # w has shape (2,)
print(v.view(3, 1) * w)
```

```
tensor([[ 4,  5],
        [ 8, 10],
        [12, 15]])
```

- Explanation:

1	1
2	2
3	3

(3,1) ⇨ (3,2)

*

4	5
4	5
4	5

(2,) ⇨ (1,2) ⇨ (3,2)

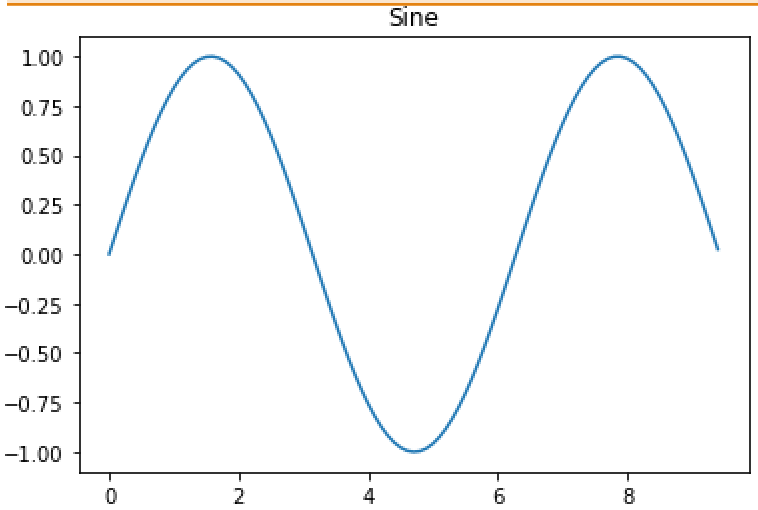
=

4	5
8	10
12	15

Plotting

- Example

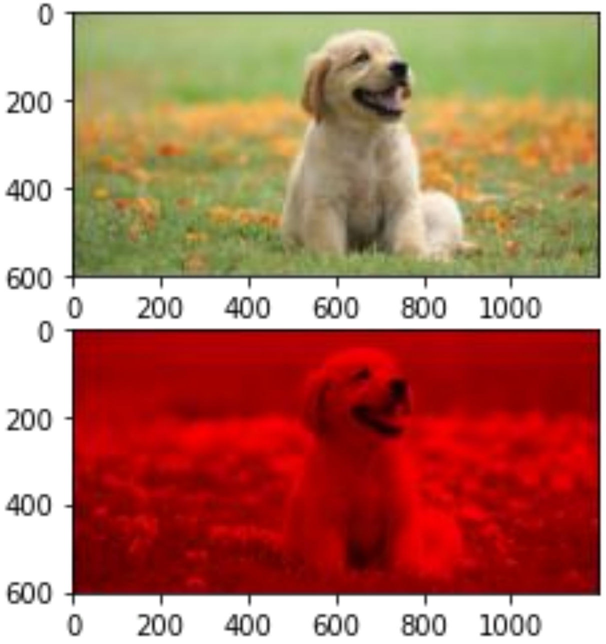
```python
import matplotlib.pyplot as plt
import torch
x = torch.arange(0,3*torch.pi,0.1)
y = torch.sin(x)

# Plot the points using matplotlib
plt.plot(x, y)
plt.title('Sine')
plt.show()  # You must call plt.show() to make graphics appear.
```



Plotting

- Example

```
import torch
import imageio
import matplotlib.pyplot as plt

img = imageio.imread('dog.jpg')
img_tinted = torch.tensor(img*[1, 0, 0] , dtype=torch.uint8)

# Show the original image
plt.subplot(2, 1, 1)
plt.imshow(img)

# Show the tinted image
plt.subplot(2, 1, 2)

# A slight gotcha with imshow is that it might give strange results
# if presented with data that is not uint8. To work around this, we
# explicitly cast the image to uint8 before displaying it.
plt.imshow(img_tinted)
plt.show()
```



0
200
400
600
0
200
400
600
800
1000
0
200
400
600
0
200
400
600
800
1000