Theano code and output

```
import theano.tensor as T
#Python program showing subtraction of two scalars
import theano
from theano import tensor
#declaring variables
a=tensor.dscalar()
b=tensor.dscalar()
#subtracting
res=a-b
#converting it to a callable object so that it takes matrix as parameters
func=theano.function([a,b],res)
#calling function
assert 20.0==(func(30.5,10.5))
#Python programming showing addition of two scalars
#Addition of two scalars
import numpy
import theano.tensor as T
from theano import function
#Declaring two variables
x=T.dscalar('x')
y=T.dscalar('y')
#summing up the two numbers
z=x+y
#converting it to a callable object so that it takes matrix as parameters
f=function([x,y],z)
f(5,7)
```

OUTPUT:

```
array(12.0)
```

```
#python program showing addition of two matrices
#Adding two matrices
import numpy
import theano.tensor as T
from theano import function
x=T.dmatrix('x')
y=T.dmatrix('y')
z=x+y
f=function([x,y],z)
f([[30,50],[2,3]],[[60,70],[3,4]])
```

OUTPUT:

```
array([[ 90., 120.],
      [  5.,   7.]])
```

```
#python program to illustrate logistic
#sigmoid function using theano
#load theano library
import theano
from theano import tensor
#declaring variables
a=tensor.dmatrix('a')
#sigmoid function
```

```
sig=1/(1+tensor.exp(-a))
#Now it takes matrix as parameters
log=theano.function([a],sig)
#calling function
print (log([[0,1],[-1,-2]]))
```

OUTPUT:

```
[[0.5        0.73105858]
[0.26894142 0.11920292]]
```

Pytorch codes and outputs:

Shape and Rank of tensors:

```
#importing torch
import torch
#creating tensors
t1=torch.tensor([1,2,3,4])
t2=torch.tensor([[1,2,3,4],[5,6,7,8],[9,10,11,12]])
#printing tensors:
print ("Tensor t1:\n",t1)
print("\n Tensor t2:\n",t2)
#Rank of tensors:
print("Rank of t1:",len(t1.shape))
print("Rank of t2:",len(t2.shape))
#Sahape of tensors:
print("Rank of t1:",t1.shape)
print("Rank of t2:",t2.shape)
```

OUTPUT:

```
Tensor t1:
 tensor([1, 2, 3, 4])

Tensor t2:
 tensor([[ 1,  2,  3,  4],
        [ 5,  6,  7,  8],
        [ 9, 10, 11, 12]])
Rank of t1: 1
Rank of t2: 2
Rank of t1: torch.Size([4])
Rank of t2: torch.Size([3, 4])
```

Creating tensor in PyTorch:

```
#importing torch module
import torch
import numpy as np
#list of values to be stored as tensor
data1 =[1,2,3,4,5,6]
data2 =np.array([1.5,3.4,6.8,9.3,7.0,2.8])
#creating tensors and printing
t1=torch.tensor(data1)
t2=torch.Tensor(data1)
t3=torch.as_tensor(data2)
```

```
t4=torch.from_numpy(data2)
print("Tensor: ",t1, "Data type: ", t1.dtype,"\n")
print("Tensor: ",t2, "Data type: ", t2.dtype,"\n")
print("Tensor: ",t3, "Data type: ", t3.dtype,"\n")
print("Tensor: ",t4, "Data type: ", t4.dtype,"\n")
```

OUTPUT:

```
Tensor:  tensor([1, 2, 3, 4, 5, 6]) Data type:  torch.int64

Tensor:  tensor([1., 2., 3., 4., 5., 6.]) Data type:  torch.float32

Tensor:  tensor([1.5000, 3.4000, 6.8000, 9.3000, 7.0000, 2.8000], dtype=torch.float64) Data type:  torch.float64

Tensor:  tensor([1.5000, 3.4000, 6.8000, 9.3000, 7.0000, 2.8000], dtype=torch.float64) Data type:  torch.float64
```

Restructuring Tensors:

```
# import torch module
import torch
# defining tensor
t = torch.tensor([[1, 2, 3, 4],[5, 6, 7, 8],[9, 10, 11, 12]])
# reshaping the tensor
print("Reshaping")
print(t.reshape(6, 2))
# resizing the tensor
print("\nResizing")
print(t.resize(2, 6))
# transposing the tensor
print("\nTransposing")
print(t.transpose(1,0))
```

OUTPUT:

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

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

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

Mathematical operations on tensors in Pytorch:

```
# import torch module
import torch
# defining two tensors
```

```
t1 = torch.tensor([1, 2, 3, 4])
t2 = torch.tensor([5, 6, 7, 8])
# adding two tensors
print("tensor2 + tensor1")
print(torch.add(t2, t1))
# subtracting two tensor
print("\ntensor2 - tensor1")
print(torch.sub(t2, t1))
# multiplying two tensors
print("\ntensor2 * tensor1")
print(torch.mul(t2, t1))
# diving two tensors
print("\ntensor2 / tensor1")
print(torch.div(t2, t1))
```

OUTPUT:

```
tensor2 + tensor1
tensor([ 6,  8, 10, 12])

tensor2 - tensor1
tensor([4, 4, 4, 4])

tensor2 * tensor1
tensor([ 5, 12, 21, 32])

tensor2 / tensor1
tensor([5.0000, 3.0000, 2.3333, 2.0000])
```

Pytorch Modules:

1)Autograd Module:

```
# importing torch
import torch
# creating a tensor
t1=torch.tensor(1.0, requires_grad = True)
t2=torch.tensor(2.0, requires_grad = True)
# creating a variable and gradient
z=100 * t1 * t2
z.backward()
# printing gradient
print("dz/dt1 : ", t1.grad.data)
print("dz/dt2 : ", t2.grad.data)
```

OUTPUT:

```
dz/dt1 :  tensor(200.)
dz/dt2 :  tensor(100.)
```