```
import torch
```

Back-propagation on one step

```
x = torch.tensor(2.0, requires_grad=True)
```

```
y = 2*x**4 + x**3 + 3*x**2 + 5*x + 1
print(y)
#Since  y  was created as a result of an operation, it has an associated gradier
#The calculation of  y  is done as:
#y=2(2)^4+(2)^3+3(2)^2+5(2)+1=32+8+12+10+1=63
#This is the value of y  when  x=2 .
```

```
tensor(63., grad_fn=<AddBackward0>)
```

```
y.backward()
```

```
print(x.grad)
#The computation is the result of
#y′=8(2)^3+3(2)^2+6(2)+5=64+12+12+5=93
#This is the slope of the polynomial at the point  (2,63) .
```

```
tensor(93.)
```

Back-propagation on multiple steps (involving layers y and z between x and our output layer out .)

```
x = torch.tensor([[1.,2,3],[3,2,1]], requires_grad=True)
print(x)
```

```
tensor([[1., 2., 3.],
        [3., 2., 1.]], requires_grad=True)
```

```
#First layer y=3*x+2
y = 3*x + 2
print(y)
```

```
tensor([[ 5.,  8., 11.],
        [11.,  8.,  5.]], grad_fn=<AddBackward0>)
```

```
#Second layer z=2*y^2
z = 2*y**2
print(z)
```

```
tensor([[ 50., 128., 242.],

        [242., 128.,  50.]], grad_fn=<MulBackward0>)
```

```
#Matrix mean to out
```

```
out = z.mean()
print(out)
```

```
tensor(140., grad_fn=<MeanBackward0>)
```

```
#Back-propagation to find the gradient of x w.r.t out
out.backward()
print(x.grad)
```

```
tensor([[10., 16., 22.],
        [22., 16., 10.]])
```