**with 类() as变量：**

(1)with后面的类()返回对象的\_\_enter\_\_()方法被调用，这个方法的返回值将被赋值给as后面的变量

(2)当with后面的代码块全部被执行完之后，将调用前面返回对象的\_\_exit\_\_()方法

真实的返回对象可能在内存中，无法看到

class Sample:

def \_\_enter\_\_(self):

print("In \_\_enter\_\_()")

return "Foo"

def \_\_exit\_\_(self, type, value, trace):

print("In \_\_exit\_\_()")

def \_\_init\_\_(self):

print("In \_\_init\_\_()")

self.name = "xx"

with Sample() as sample:*#执行顺序：\_\_init\_\_()—>\_\_enter\_\_()—>with语句后面的语句块—>\_\_exit\_\_()*

print(sample)

>>>

In \_\_init\_\_()

In \_\_enter\_\_()

Foo

In \_\_exit\_\_()

**with open("123.txt") as f:**

自动关闭文件

f.open()自动调用了\_\_exit\_\_方法，f.\_\_exit\_\_()自动调用了close方法

**with 类()：**

(1)with后面的类()返回对象的\_\_enter\_\_()方法被调用

(2)当with后面的代码块全部被执行完之后，将调用前面返回对象的\_\_exit\_\_()方法

class Sample:

def \_\_enter\_\_(self):

print("In \_\_enter\_\_()")

return "Foo"

def \_\_exit\_\_(self, type, value, trace):

print("In \_\_exit\_\_()")

def \_\_init\_\_(self):

print("In \_\_init\_\_()")

self.name = "xx"

with Sample():

pass

>>>

In \_\_init\_\_()

In \_\_enter\_\_()

In \_\_exit\_\_()

**with torch.no\_grad():**

在此语句下，所有新计算得出的tensor的requires\_grad都自动设置为False，且grad\_fn也为None，一般用于不需要反向传播的场景，如验证网络

x = torch.randn(10, 5, requires\_grad = True)

y = torch.randn(10, 5, requires\_grad = True)

z = torch.randn(10, 5, requires\_grad = True)

with torch.no\_grad():

w = x + y + z

print(w.requires\_grad)

print(w.grad\_fn)

print(w.requires\_grad)

>>>

False

None

False