In [1]:

```
import torch
import torch.nn as nn
import numpy as np
```

In [4]:

```
class DoorLock(nn.Module):
    def __init__(self):
        super().__init__()
        self.f = nn.Sequential(nn.Linear(100, 1), nn.Sigmoid())
        for p in self.f.parameters():
            p.requires_grad = False

    def forward(self, x):
        y = self.f(x)
        if(y > 0.9):
            print('Opened!')
        return y
```

In [5]:

```
class DoorHack(nn.Module):
    def __init__(self, locker):
        super().__init__()
        self.g = nn.Sequential(nn.Linear(100, 100),)
        self.locker = locker

    def forward(self, z):
        y = self.locker(self.g(z))
        return y
```

In [6]:

```
num_trials = 50
locker = DoorLock()
hacker = DoorHack(locker)
```

In [7]:

```
z = torch.randn(1, 100)

optimizer = torch.optim.SGD(hacker.g.parameters(), lr=0.1)
losses = []

for i in range(num_trials):
  optimizer.zero_grad()
  output = hacker.forward(z)
  loss = torch.mean((output - 1)**2)
  loss.backward()
  optimizer.step()
  losses.append(loss.item())
```

```
Opened!
Opened!
Opened!
Opened!
Opened!
Opened!
Opened!
Opened!
Opened!
Opened!
Opened!
Opened!
Opened!
Opened!
Opened!
Opened!
```

```
Opened!
Opened!
Opened!
Opened!
Opened!
Opened!
Opened!
Opened!
Opened!
Opened!
Opened!
Opened!
Opened!
Opened!
Opened!
Opened!
Opened!
Opened!
Opened!
Opened!
Opened!
Opened!
Opened!
Opened!
Opened!
Opened!
Opened!
Opened!
Opened!
Opened!
```