The simplest (and trivial) Markov Chain Monte Carlo code with the Metropolis-Hastings algorithm.

Sampling from a target distribution $\pi(x)$ = univariate gaussian with $\mu = 1$ and $\sigma = 2$.

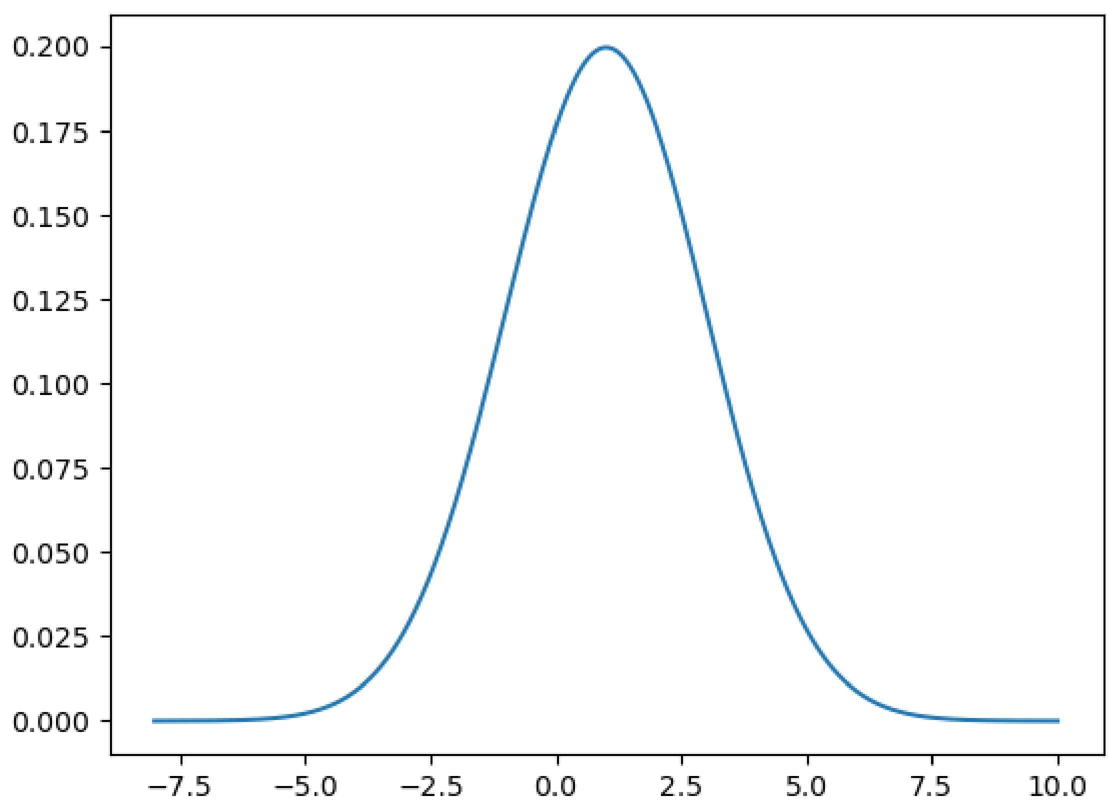
In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats as stats
```

This is the target distribution $\pi(x)$. Note that this does **not** need to be normalized (try it)!

In [2]:
```
def targetPDF(x):
    return stats.norm.pdf(x, loc=1, scale=2)
```

In [3]:
```
xval = np.linspace(-8,10,1000)
pval = targetPDF(xval)
plt.plot(xval, pval)
plt.show
```

Out[3]: `<function matplotlib.pyplot.show(close=None, block=None)>`



In [39]:
```
np.random.seed(2384)
N     = 100000        # number of candidate steps
x0    = 50            # first guess (well outside)
xstep = 3.0           # step size (note that there are good and bad choices for
p     = targetPDF(x)  # target \pi(x)
```

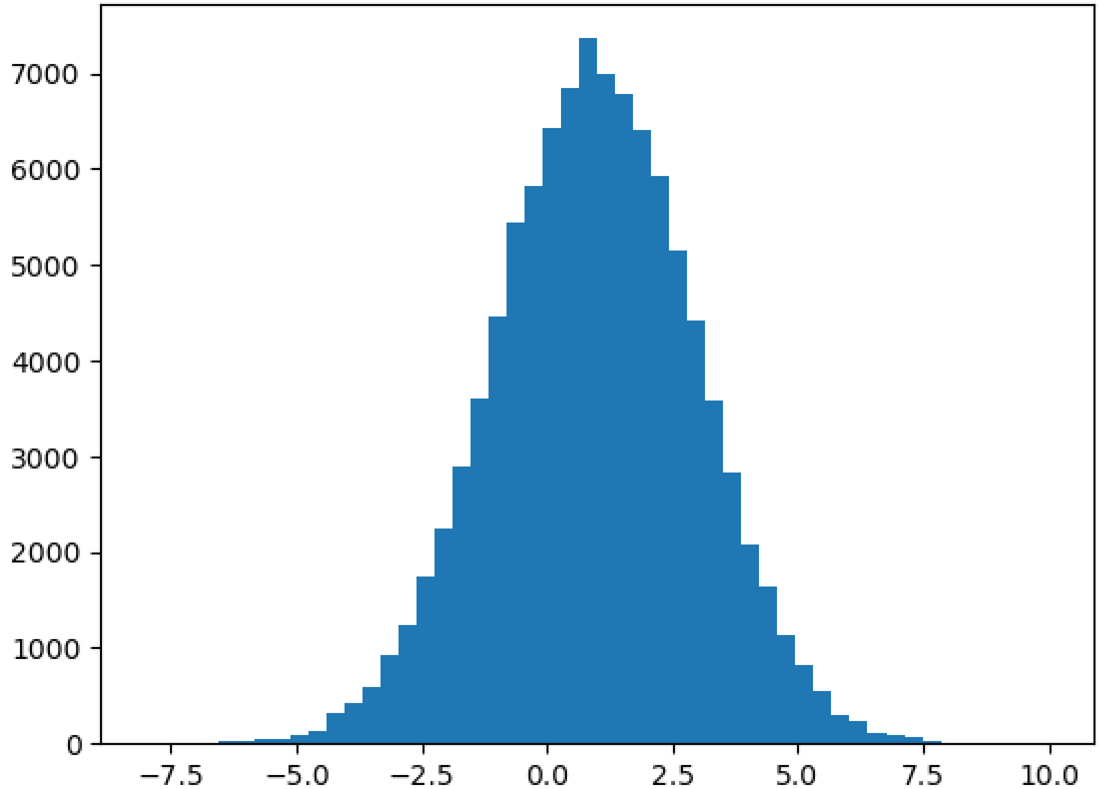
In [40]:
```
#write this function
#def MCMC(targetPDF, N, x0, xstep):
#    return samples
```

In [41]:
```
def MCMC(pifunc, N, x0, xstep):

    noyes   = np.zeros(2, dtype=int)  # to keep track of reject and accept frac
    xold    = x0                      # xold
    pold    = pifunc(xold)            # pi(xold)
    samples = []                      # the MCMC chain

    for i in range(N):
        xnew = xold + xstep * np.random.normal()
        pnew = pifunc(xnew)
        ratio = pnew/pold

        if ratio >= 1.0:
            take_step = 1
        else:
            stepran = np.random.uniform()
            if stepran < ratio:
                take_step = 1
            else:
                take_step = 0

        noyes[take_step] += 1    # index 0,1 is number rejected, accepted
        if take_step == 1:
            pold = pnew
            xold = xnew

        samples.append(xold)

    samples = np.array(samples)

    print("reject / accept fractions = ", noyes/N)

    return samples
```

In [42]:
```
samples = MCMC(targetPDF, N, x0, xstep)
```

```
reject / accept fractions =  [0.40969 0.59031]
```

In [43]:
```
xhist = plt.hist(samples, range=[-8,10], bins=50)
```



In [44]:
```
#for i in np.arange(N):
#    print(samples[i])
```

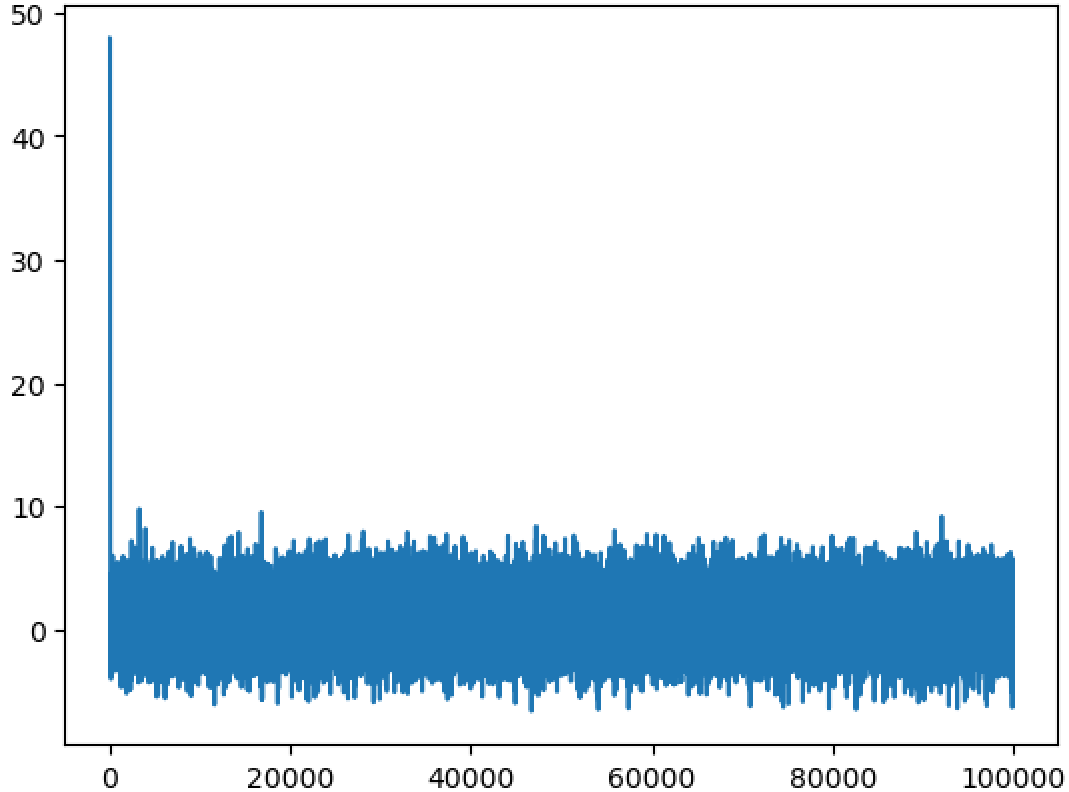
In [45]:
```
print(np.mean(samples), np.std(samples))
```

```
0.9710612327846642 2.0705611428615343
```

In [46]:
```
steps = np.arange(N)
```

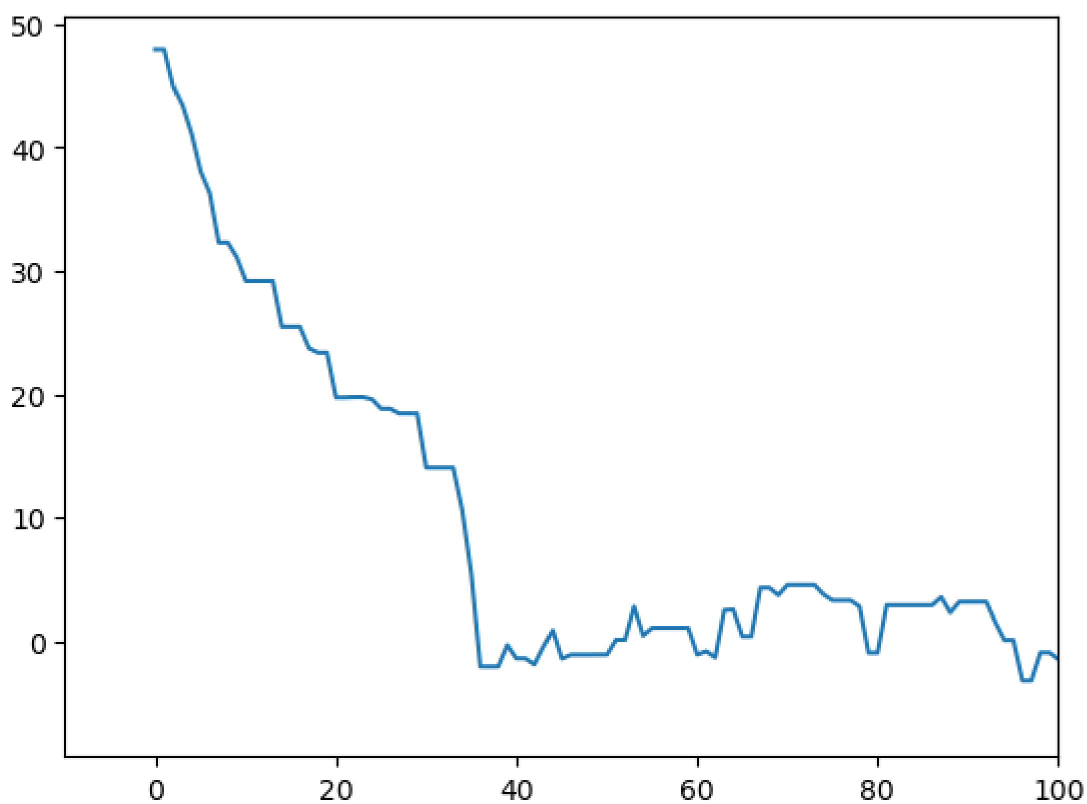
In [53]:
```
plt.plot(steps, samples)
plt.show
```

Out[53]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```



In [52]:
```
plt.plot(steps, samples)
plt.xlim([-10,100])
plt.show
```

Out[52]: <function matplotlib.pyplot.show(close=None, block=None)>



In [54]:
```
def targetPDF2(x, y):
```

```
    g1 = stats.multivariate_normal.pdf((x,y), [-1,-1], [[1,-0.8],[-0.8,1]])
    g2 = stats.multivariate_normal.pdf((x,y), [1,2], [[1.5,0.6],[0.6,0.8]])
    return g1+g2
```

In [56]:

```
np.random.seed(2384)
N     = 100000          # number of candidate steps
A0    = [0,0]           # first guess (well outside)
step  = [1.0, 1.0]      # step size (note that there are good and bad choices
p     = targetPDF2(*A0) # target \pi(x)
```

In [57]:

```
p
```

Out[57]:

```
0.014441725137053189
```

In [ ]: