In [1]:

```
import matplotlib.pyplot as py
import numpy as np
```

$M_{pl} = 1$, potentials $V = V_0\phi^k$, $H_0 = 2.5 \times 10^{-5}$ simulating H trajectories using HJ formalism, so $\frac{dH}{d\phi} = \sqrt{1.5H^2 - 0.5V}$

In [13]:

```
def trajectory(V_0, k, phi_0, dphi = 0.0001, boundmultiplier = 100):
    T =  int(phi_0/dphi)
    phi = [phi_0 - dummy for dummy in np.linspace(0, phi_0, boundmultiplier*T+1)]   #mesh for phi
    H = np.zeros(boundmultiplier*T+1)
    H[0]  = 2.5e-5
    epsilon = 0
    n = 0
    while ((epsilon < 1) and (n < boundmultiplier*T)):
        K1 = (1.5*(H[n]**2) - 0.5*V_0*(phi[n])**k)**0.5
        if not isinstance(K1, float):
            return phi[0:n], H[0:n], n, epsilon
        K2 = (1.5*((H[n] - 0.5*dphi*K1)**2) - 0.5*V_0*(phi[n] - 0.5*dphi)**k)**0.5
        if not isinstance(K2, float):
            return phi[0:n], H[0:n], n, epsilon
        K3 = (1.5*((H[n] - 0.5*dphi*K2)**2) - 0.5*V_0*(phi[n] - 0.5*dphi)**k)**0.5
        if not isinstance(K3, float):
            return phi[0:n], H[0:n], n, epsilon
        K4 = (1.5*((H[n] - dphi*K3)**2) - 0.5*V_0*(phi[n] - dphi)**k)**0.5
        if not isinstance(K4, float):
            return phi[0:n], H[0:n], n, epsilon
        grad = (K1 + 2*K2 + 2*K3 + K4)/6
        H[n+1] = H[n] - dphi*grad
        if (not isinstance((1.5*(H[n+1]**2) - 0.5*V_0*(phi[n+1])**k)/(H[n+1]**2), float)) or (not (1.5*(H[n+1]**2) - 
0.5*V_0*(phi[n+1])**k)/(H[n+1]**2) < 0.5):
            break
        epsilon = 2*(1.5*(H[n+1]**2) - 0.5*V_0*(phi[n+1])**k)/(H[n+1]**2)
        n += 1
    return phi[0:n+1], H[0:n+1], n, epsilon
```

In [12]:

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
<ipython-input-12-0ade25c19a2c> in <module>
----> 1 isinstance(1/0, float)

ZeroDivisionError: division by zero
```

In [8]:

Out[8]:

numpy.float64

In [14]:

```
242**0.5
```

Out[14]:

15.556349186104045

In [12]:

```
(2.5e-5)**2/(242*3) #Slow roll V_0
```

Out[12]:

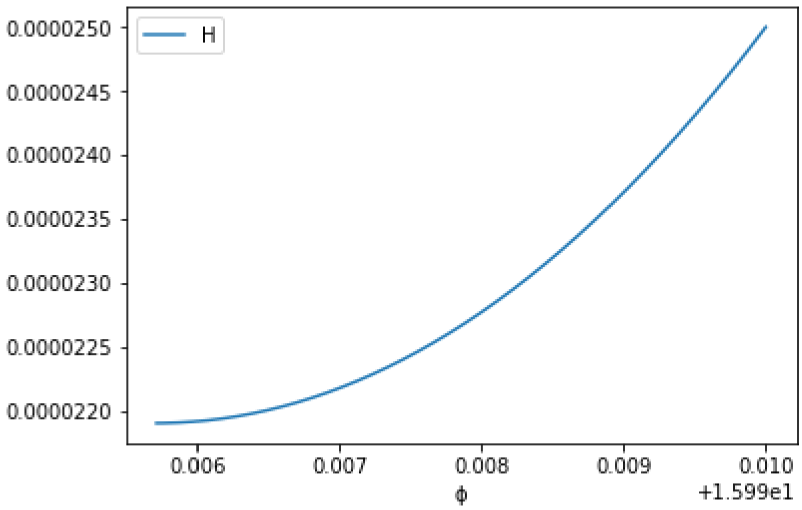
8.608815426997245e-13

In [14]:

```
phi, H, n, epsilon = trajectory(9e-11, 1, 16)
py.plot(phi, H, label='H')
py.xlabel('ϕ')
py.legend()
print("n =", n, end = '\n')
print("ε =", epsilon)
```

```
C:\Users\Jacob\Anaconda3\lib\site-packages\ipykernel_launcher.py:9: RuntimeWarning: inva
lid value encountered in double_scalars
  if __name__ == '__main__':
```

```
n = 4286
ε = -2.5155078675313932e-08
```



In [4]:

```
from tabulate import tabulate
```

In [8]:

```
#data = []
#for num in range(10, 20):
#    nlast = False
#    for x in [1.0 + n for n in np.linspace(0, 0.9,91)]:
#        _1, _2, n, _3 = trajectory(x*1.0e-10, 1, phi_0 = num)
#        if n == 1:
#            if nlast == True:
#                data.append([num, x])
#                break
#            else:
#                continue
#        nlast = True

#print(tabulate(data))
```

```
C:\Users\Jacob\Anaconda3\lib\site-packages\ipykernel_launcher.py:9: RuntimeWarning: inva
lid value encountered in double_scalars
  if __name__ == '__main__':

--  ----
10  1.88
11  1.71
12  1.57
13  1.45
14  1.34
15  1.26
16  1.18
17  1.11
18  1.05
--  ----
```

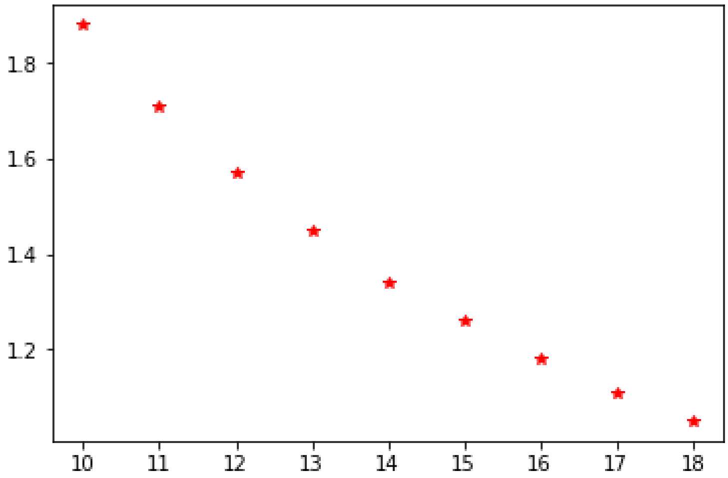
In [16]:

```
#import json
#with open('v0_and_phi_table.txt', 'w') as f:
#    f.write(json.dumps(data))

#Now read the file back into a Python list object
with open('v0_and_phi_table.txt', 'r') as f:
    table = json.loads(f.read())

print(tabulate(table, headers = ["ϕ_60","A good value of V_0 x 10^10"]))
```

```
  ϕ_60    A good value of V_0 x 10^10
------  -----------------------------
    10                           1.88
    11                           1.71
    12                           1.57
    13                           1.45
    14                           1.34
    15                           1.26
    16                           1.18
    17                           1.11
    18                           1.05
```

In [18]:

```
for x in table:
    py.plot(x[0], x[1], 'r*')
```



In [ ]:

```

```

We had been choosing random initial ϕ_0. Now we will check corresponding N values integrate backwards while $N < N_{\text{start}} \approx 60$

$N = \int_{\phi_0}^{\phi_N} \sqrt{\frac{2}{\varepsilon}} d\phi$

In [ ]:

```

```

In [23]:

```
def init_phi(Nstart, V_0, k, phi_0 = 10, dphi = 0.0001, boundmultiplier = 100):
    T = int(phi_0/dphi)
    phi, J, _, epsilon = trajectory(V_0, k, phi_0, dphi, boundmultiplier)
    N = 0
    phi = phi[-1]
    H = J[-1]
    ε = np.zeros(boundmultiplier*T + 1)
    ε[0] = epsilon
    n = 0
    while (N < Nstart) and (n < boundmultiplier*T):
        K1 = (1.5*(H**2) - 0.5*V_0*(phi)**k)**0.5
        K2 = (1.5*((H + 0.5*dphi*K1)**2) - 0.5*V_0*(phi + 0.5*dphi)**k)**0.5
        K3 = (1.5*((H + 0.5*dphi*K2)**2) - 0.5*V_0*(phi + 0.5*dphi)**k)**0.5
        K4 = (1.5*((H + dphi*K3)**2) - 0.5*V_0*(phi + dphi)**k)**0.5
        grad = (K1 + 2*K2 + 2*K3 + K4)/6
        H += dphi*grad
        if not isinstance((1.5*(H**2) - 0.5*V_0*(phi+dphi)**k)/(H**2), float):
            break
        ε[n+1] = 0.5*(1.5*(H**2) + 0.5*V_0*(phi+dphi)**k)/(H**2)
        #SPACE
        J1 = (2 / ε[n])**0.5
        J2 = (2 / ε[n+1])**0.5
        N += dphi*(J1 + J2)/4
        n += 1
        phi += dphi
    return phi, N, H
```

In [22]:

```
init_phi(60, 0, 1)
```

Out[22]:

(46.74239900043813, 60.000047125118414, 873307422808083.2)

We have been choosing $H_0 = 2.5 \times 10^{-5}$. If we want to continue this, we need to scale the Hamilton Jacobi equation so that $V_0 \to \frac{V_0}{H_0^2}$ or change H_0

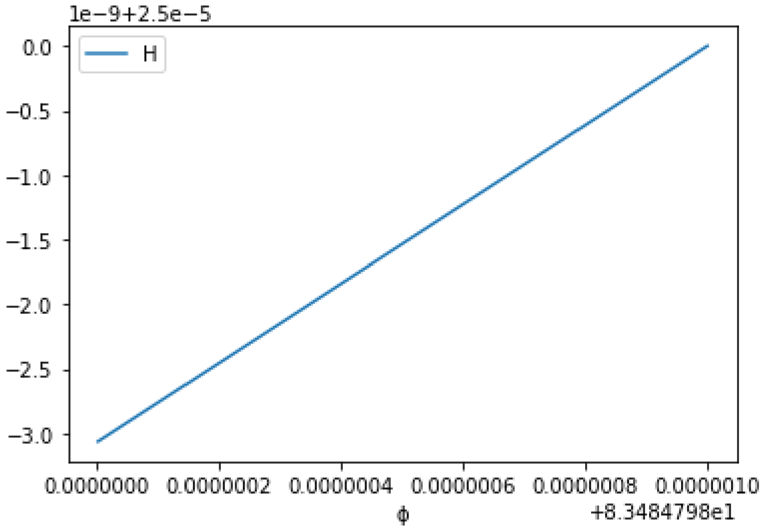
In [24]:

```
def tot_traj(Nstart, V_0, k, phi_0 = 10, dphi = 0.0001, boundmultiplier = 100):
    initphi, _, H0 = init_phi(Nstart, V_0, k, phi_0, dphi, boundmultiplier)
    phi, H, n, epsilon = trajectory(V_0/(H0**2), k, initphi, dphi, boundmultiplier)
    return phi, H, n, epsilon
```

In [25]:

```
phi, H, n, epsilon = tot_traj(60, 0, 1)
py.plot(phi, H, label='ϕ')
py.xlabel('ϕ')
py.legend()
print("n =", n, end = '\n')
print("ε =", epsilon)
```

n = 1
ε = 3.0



I now try to narrow down some sensible ϕ_0 values

In [ ]:

```
S = []
for x in [2.3 + n for n in np.linspace(-0.1, 0.1, 101)]:
    _1, _2, n, _3 = tot_traj(60, 1, 1, phi_0 = x)
    S.append(n)
m = max(S)
for i, x in enumerate(S):
    if x == m:
        print(i)
        break
```

In [ ]:

In [ ]:

In [ ]: