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

In [2]:
```python
# Monte Carlo method for path-dependent options
def fslb_put(S0,K,T,r,sigma,N,M):
    dt = T/N
    nu = r - 0.5*sigma**2
    A = np.zeros(M);
    for j in range(M):
        S = np.zeros(N+1)
        S[0] = S0
        for i in range(N):
            Z = sp.norm.rvs(); # random var from standard normal dist
            S[i+1] = S[i]*np.exp(nu*dt + sigma*np.sqrt(dt)*Z); # asset mod
el
        S_min = np.min(S) # = (1/T)*(sum S_i*dt) = (1/N)*(sum S_i)
        A[j] = np.exp(-r*T)*max(K-S_min,0);
    aM = np.mean(A);
    bM = np.sqrt(np.var(A,ddof=1));
    #plt.plot(S)
    #plt.hlines(S_avg,0,N)
    #plt.show()
    #print('euro payout  =',max(S[N]-K,0))
    #print('asian payout =',max(S_avg-K,0))
    return [aM, bM];

# functions for exact value of European Put for comparison
def d1(S,K,T,t,r,sigma):
    d1 = (np.log(S/K) + (r+0.5*sigma**2)*(T-t))/(sigma*np.sqrt(T-t))
    return d1
def d2(S,K,T,t,r,sigma):
    d2 = (np.log(S/K) + (r-0.5*sigma**2)*(T-t))/(sigma*np.sqrt(T-t))
    return d2
def P_euro(S,K,T,t,r,sigma):
    P_euro = K*np.exp(-r*(T-t))*sp.norm.cdf(-d2(S,K,T,t,r,sigma))-\
        S*sp.norm.cdf(-d1(S,K,T,t,r,sigma))
    return P_euro
```

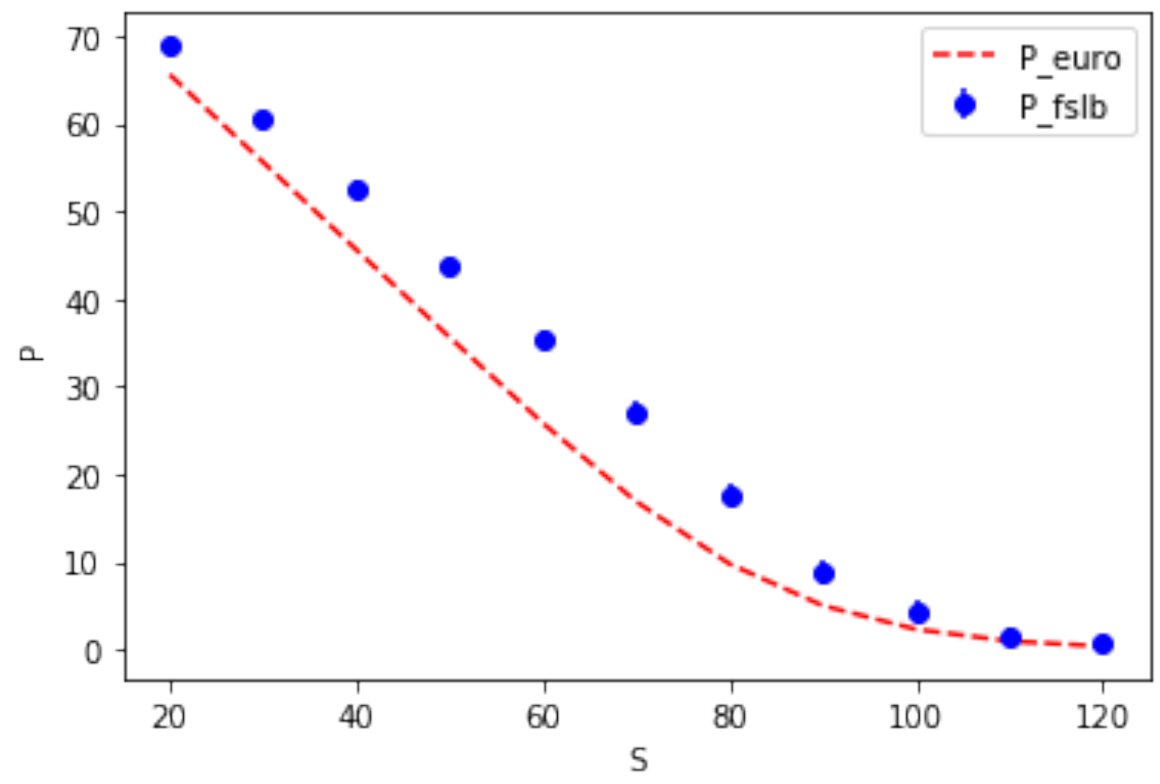
In [3]:
```python
# find option value using Monte Carlo
S0= 80
r = 0.05
sigma = 0.2
t = 0
T = 1
K = 90

# simulation parameters
M = 100 # number of Monte Carlo simulations
N = 50 # number of time steps for asset model

[A, stddev] = fslb_put(S0,K,T,r,sigma,N,M)
Peuro = P_euro(S0,K,T,t,r,sigma)
print('fixed-strike lookback put = ',A,'+/-',1.96*stddev/np.sqrt(M))
print('P_euro =',Peuro)
```

```
fixed-strike lookback put =  17.35897136094899 +/- 1.2401717709832432
P_euro = 9.77504035241828
```

In [4]:
```python
# set up array to find value as function of S
nfine = 11
Smin = 20
Smax = 120
S = np.linspace(Smin,Smax,nfine)
aM = np.zeros(nfine)
a_err = np.zeros(nfine)
Peuro = np.zeros(nfine)
for k in range(nfine):  # loop over all S values
    [aM[k],bM] = fslb_put(S[k],K,T,r,sigma,N,M)
    a_err[k] = 1.96*bM/np.sqrt(M)
    Peuro[k] = P_euro(S[k],K,T,t,r,sigma)
plt.errorbar(S,aM,yerr=a_err,fmt='bo',label='P_fslb')
plt.plot(S,Peuro,'r--',label='P_euro')
plt.legend()
plt.xlabel('S')
plt.ylabel('P')
plt.show()
```



P_fslb generally higher than P_euro because S_T has a lognormal distribution with fat tail for large S, while S_min includes small t values for which there is only small deviations from S0

In [ ]: