In [1]:
```
# Lab Book 01
def LabBook01(n):
    sum = 0
    for x in range(0, 2 * n, 2):
        sum += x
    return sum


n = 20
print(LabBook01(n))
print(n * (n - 1))
```

```
380
380
```

In [2]:
```
# Lab Book 02
def ex(x):
    """
    Calculate the exponential e^x using a Taylor series.
    This is a docstring: it can have as much information as you wan
    Usually, this means at least explaining what the function does,
    what input(s) it expects, and what output(s) it has.
    Any limitations/restrictions on inputs should go here (for exam
    'this only works if x > 0').
    """
    ans = 0
    n=1
    term = 1
    while ans + term != ans:
        ans = ans + term
        term = term * (x / n)
        n=n+1
    return ans

print(ex(1))
print(ex(10))
print(ex(-10))
```

```
2.7182818284590455
22026.465794806714
4.539992967040021e-05
```

Lab Book 03 The loop will stop as long as term converges to zero and in this case means different iteration times based on diffenent input x.

In [3]:
```
# Lab Book 04
mylist = [1, 4, 9, 16, 25, 36, 49]
print(mylist[1:len(mylist)-1])
```

```
[4, 9, 16, 25, 36]
```

In [4]:
```
# Lab Book 05
import math
import numpy as np
diff = np.zeros(40)
for i in range(-20,20,1):
    diff[i] = ex(i) - math.exp(i)
print(diff)
```

```
[ 0.00000000e+00  4.44089210e-16 -1.77635684e-15 -7.10542736e-15
  2.84217094e-14  0.00000000e+00  0.00000000e+00 -6.82121026e-13
 -1.81898940e-12 -1.81898940e-12 -3.63797881e-12  1.45519152e-11
 -1.16415322e-10 -5.82076609e-11  4.65661287e-10  0.00000000e+00
 -1.86264515e-09  1.11758709e-08  0.00000000e+00 -2.98023224e-08
  4.08640821e-09 -1.34614048e-09  7.53740589e-10 -6.10385327e-11
 -6.01415855e-11  7.68208022e-12  1.54589755e-11  1.01043208e-12
 -1.83593900e-12  4.61436058e-13 -9.20846448e-14 -1.87626173e-13
 -4.47788508e-14 -4.76224962e-15  3.71013983e-15 -8.29197822e-16
 -2.25514052e-16  8.32667268e-17  2.77555756e-17  1.11022302e-16]
```

Lab Book 06 math.ert() or math.erfc() The meaning of erf(x) is for a random variable Y that is normally distributed with mean 0 and standard deviation 1/√2, it is the probability that Y falls in the range [-x,x].

In [5]:
```
# Lab Book 07
def phi(x):
    return (1 + math.erf(x/math.sqrt(2))) / 2
def european_call(K, S, T, r, sigma):
    d1 = (math.log1p(S/K-1) + (r+math.pow(sigma,2)/2)*T) / (sigma *
    d2 = d1 - sigma * math.sqrt(T)
    C = phi(d1) * S - phi(d2) * K * math.pow(math.exp(1),(-r*T))
    return C

print(european_call(90,100,0.03,0.5,0.1))
```

```
11.33992543572478
```

In [6]:
```
# Lab Book 08
def current(RL,V,RS):
    """
    Calculate the current given the voltage source with voltage V
    and internal resistance RS supplying a load of resistance RL.
    Input: RL = load of resistance (list of floating-point numbers)
         : V = Voltage (floating-point number)
         : RS = internal resistance (floating-point number)
    Output: current (list of floating-point numbers)
    """
    n = len(RL)
    I=n*[0.0] #I is a list with the same size as RL
    for j in range(n):
        I[j] = V / (RL[j] + RS)
    return I
RL = list(range(1,6))
I = current(RL, 1.0, 0.0)
print(I)
```

```
[1.0, 0.5, 0.3333333333333333, 0.25, 0.2]
```

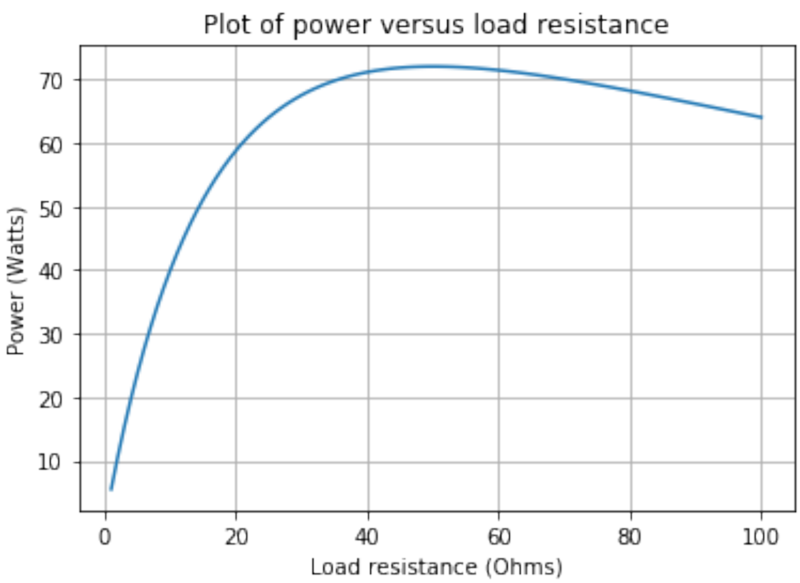
Lab Book 08 I think it is right.

In [7]:
```
# Lab Book 09
def power(I, RL):
    PL = (np.array(I) ** 2) * RL
    return PL
print(power(I,RL))
```

```
[1.         0.5        0.33333333 0.25       0.2       ]
```

In [8]:
```
import matplotlib.pyplot as plt
def power_plot(RL, PL):
    """
    Plot the power PL versus the load of resistance RL.
    Input: RL = load of resistance (list)
         : PL = power (list)
    Output: none
    """
    plt.clf()
    plt.plot(RL, PL)
    plt.title('Plot of power versus load resistance')
    plt.xlabel('Load resistance (Ohms)')
    plt.ylabel('Power (Watts)')
    plt.grid()
    plt.show()
    return

RS = 50.0
# Set the voltage source to 120V
V = 120.0
# Create a list of possible values for the load resistance
RL = list(range(1, 101))
# Calculate the current and power
I = current(RL, V, RS)
PL = power(I, RL)
# Plot the power versus load resistance
power_plot(RL, PL)
```



In [9]:
```
# Lab Book 10
def optimisePower(RL,PL):
    PLMax = np.max(PL)
    RLMax = RL[np.where(PL == PLMax)[0][0]]
    return RLMax, PLMax
RLMax, PLMax = optimisePower(RL,PL)
print(RLMax)
print(PLMax)
```

```
50
72.0
```

In [ ]: