In [1]:
```
import sympy as sp
from sympy.plotting import (plot, plot_parametric,plot3d_parametric_surface, plot3d_parametric_line,plot3d)
```

INTRODUCTION TO SEQUENCES AND SERIES IN PYTHON

Sequences can be thought of as functions whose domain is the set of nonnegative integers (or a subset thereof). So while symbolic manipulations can be done on them, plotting requires a different plot command found in the matplotlib.pyplot library. Since the command is also "plot", we have to distinguish it from the symbolic plot command. One way is to call the full library path, i.e.,

matplotlib.pyplot.plot(...)

As you can see, this can get rather cumbersome. So, in typical mathematician fashion, we create a shorthand notation for it.

In [2]:
```
import matplotlib.pyplot as plt
```

We can now use the numerical plot command as plt.plot.

NOTE: The example here (and in future overviews) is NOT a copy/paste to solve the problems in lab. However, it will USE many of the features you will use to solve your problems, such as (in this case) listing sequences and partial sums, plotting sequences and partial sums, summing series, and using for loops to create recursive sequences.

EXAMPLE 1:

1. Given the series sum(1/n, n=1..oo)

   a) List the first ten terms of the series and the first ten partial sums

   b) Plot the first 50 terms of the series and the first 50 partial sums on the same graph.

   c) Show that the limit of the terms is 0, but the limit of the partial sums diverges.

For part a), we need to use list comprehension for the terms and a command in the numpy library for the partial sums. NOTE that instead of defining the expression separately, we could just define

a1_10=[1/n for i in range(1,11)]

In [3]:
```
n=sp.symbols('n',integer=True) #NOTE that we can assume n to be an integer-and also positive if necessary
a=1/n #the sequence representing the terms of the series
a1_10=[a.subs({n:i}) for i in range(1,11)]
# Notice the "range" command above. It creates a list of integers from 1 INCLUSIVE to 11 EXCLUSIVE
print('The list of terms is',a1_10)
from numpy import cumsum #cumsum is short for "cumulative summation"-basically the Nth partial sum of the list!
s1_10=cumsum(a1_10)
print('The list of partial sums is',s1_10)
# We probably want decimal representation of the partial sums.
# You cannot evalf a list, so it requires list comprehension
sfloat=[i.evalf() for i in s1_10]
print('As floating point, the list of partial sums is',sfloat)
```

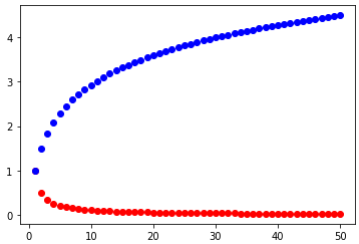
```
The list of terms is [1, 1/2, 1/3, 1/4, 1/5, 1/6, 1/7, 1/8, 1/9, 1/10]
The list of partial sums is [1 3/2 11/6 25/12 137/60 49/20 363/140 761/280 7129/2520 7381/2520]
As floating point, the list of partial sums is [1.00000000000000, 1.50000000000000, 1.83333333333333, 2.08333333333333, 2.28333333333333, 2.45000000000000, 2.59285714285714, 2.7178571
4285714, 2.82896825396825, 2.92896825396825]
```

It looks like the terms will approach 0 (fairly obvious), but the partial sums may be approaching 3? Let's check the plot in part b)-remember to use plt.plot to plot lists of values. Also note the 'o' option to plot points rather than connecting them. The r and b refer to the color, though that is mainly for your view since you are likely printing black & white.

Also, notice that multiple graphs do NOT have to be put in "tuples" unlike symbolic plots!

In [ ]:
```
nvals=range(1,51) #again note the exclusive right endpoint!
a1_50=[a.subs({n:i}) for i in nvals]
s1_50=cumsum(a1_50)
plt.plot(nvals,a1_50,'ro',nvals,s1_50,'bo')
```

Out[ ]:
```
[<matplotlib.lines.Line2D at 0x7feed2de75b0>,
 <matplotlib.lines.Line2D at 0x7feed2de7670>]
```



Obviously, the partial sums are not approaching 3. This function is simple enough that we can use the limit command for the terms and the summation command for the partial sums (remember that the limit of the partial sums is the same as the sum of the infinite series! "summation" is preferable for an infinite sum; "sum" for a finite sum.) If you had a formula for the partial sums, you can just use the limit command on it to determine the sum of the series.

In [5]:
```
# oo refers to infinity. It is a SymPy object, so you have to enter "sp.oo"
L=sp.limit(a,n,sp.oo)
S=sp.summation(a,(n,1,sp.oo))
print('The limit of the sequence of terms is',L,'but the series diverges to',S)
```

```
The limit of the sequence of terms is 0 but the series diverges to oo
```

EXAMPLE 2:

Given a(0)=11, define the sequence a recursively by

a(n) = |a(n-1)-1| if n is odd

a(n) = a(n-1)/2 if n is even

a) List the first 30 terms of the sequence. What appears to be happening?
b) Find (by hand) a recursive formula for the odd-numbered terms and a recursive formula for the even-numbered terms. Assuming the limit exists (a(n+1) --> L and a(n) --> L), find it.

We use a for loop to define the sequence. The easiest strategy is to define a as a list of one number, then append each new value to the list. NOTE in this case, the rule for appending depends on whether n is even or odd (if/else statements). In Python, the % refers to the remainder when dividing.

In [6]:
```
a=[11]
for n in range(1,31):
    if n % 2:  # true=1 which means n is odd
        a.append(abs(a[n-1]-1))
    else:  # false=0 which means n is even
        a.append(a[n-1]/2)
print('The sequence of terms is',a)
print('It appears that they eventually bounce between 2/3 and 1/3.')
L=sp.symbols('L',positive=True)
# For even terms, a(2n+2)=|a(2n)/2 - 1|.  Replace both with L
print('For even-numbered terms, the limit is',sp.solve(L-abs(L/2-1),L))
# For odd terms, a(2n+1)=|(a(2n-1)-1)/2|. Replace both with L
print('For odd-numbered terms, the limit is',sp.solve(L-abs((L-1)/2),L))
```

```
The sequence of terms is [11, 10, 5.0, 4.0, 2.0, 1.0, 0.5, 0.5, 0.25, 0.75, 0.375, 0.625, 0.3125, 0.6875, 0.34375, 0.65625, 0.328125, 0.671875, 0.3359375, 0.6640625, 0.33203125, 0.667
96875, 0.333984375, 0.666015625, 0.3330078125, 0.6669921875, 0.33349609375, 0.66650390625, 0.333251953125, 0.666748046875, 0.3333740234375]
It appears that they eventually bounce between 2/3 and 1/3.
For even-numbered terms, the limit is [2/3]
For odd-numbered terms, the limit is [1/3]
```

In [ ]: