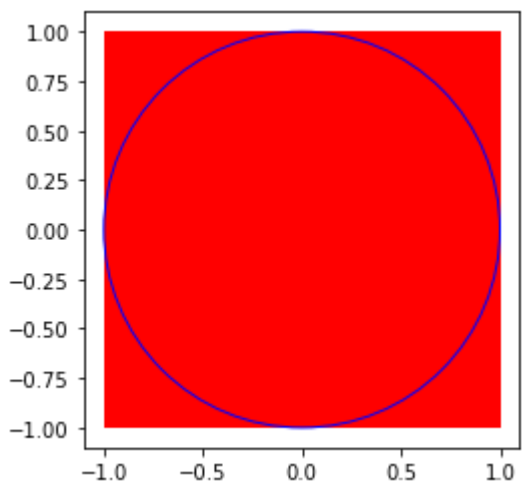
# Calculating $\pi$ using a Monte Carlo algorithm

- Börge Göbel

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

In [2]:
```
fig = plt.figure()
ax = fig.add_subplot(111, aspect='equal')

rectangle = patches.Rectangle((-1,-1),2,2, facecolor='red')
ax.add_patch(rectangle)

circle = patches.Circle((0,0),1, facecolor='none', edgecolor='blue')
ax.add_patch(circle)

plt.xlim([-1.1,1.1])
plt.ylim([-1.1,1.1])
```

Out[2]: (-1.1, 1.1)




## Comparing the areas

Area of the circle: $A_{\text{circle}} = \pi r^2 = \pi$

Area of the square: $A_{\text{square}} = a^2 = 4$

This means $\pi = 4\frac{A_{\text{circle}}}{A_{\text{square}}}$. We can use this ratio to estimate the value of $\pi$.

### Measure the area ratio by counting randomly generated points

In [3]:
```
points = 10

rand = 2*np.random.rand(2*points) -1
print(len(rand))
print(rand)
```

```
20
[-0.63761604  0.7274327  -0.30130913  0.10564772 -0.42630659 -0.74359571
 -0.50075805 -0.23860334  0.74576652  0.28601528 -0.0017804   0.2459268
 -0.30265802  0.48325214  0.25959074  0.40358026 -0.60144693  0.86410906
  0.60689826  0.01837142]
```

In [4]:
```
randpoints = rand.reshape( points,2 )
print(randpoints)
```

```
[[-0.63761604  0.7274327 ]
 [-0.30130913  0.10564772]
 [-0.42630659 -0.74359571]
 [-0.50075805 -0.23860334]
 [ 0.74576652  0.28601528]
 [-0.0017804   0.2459268 ]
 [-0.30265802  0.48325214]
 [ 0.25959074  0.40358026]
 [-0.60144693  0.86410906]
 [ 0.60689826  0.01837142]]
```

In [5]:
```
normpoints = randpoints[:,0]**2 + randpoints[:,1]**2
```

In [6]:
```
pointsOut = randpoints[normpoints > 1]
pointsIn = randpoints[normpoints <= 1]
```

In [7]:
```
piapprox = 4 * len(pointsIn)/points
print(piapprox)
```

```
3.6
```

In [8]:
```
piapprox - np.pi
```

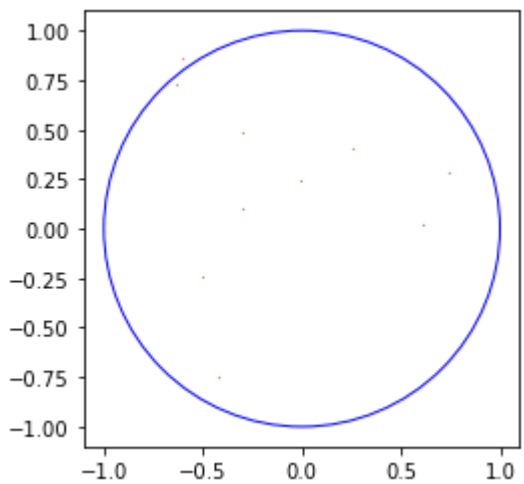
Out[8]:
```
0.458407346410207
```

In [9]:

```
fig = plt.figure()
ax = fig.add_subplot(111, aspect='equal')

#rectangle = patches.Rectangle((-1,-1),2,2, facecolor='red')
#ax.add_patch(rectangle)

circle = patches.Circle((0,0),1, facecolor='none', edgecolor='blue')
ax.add_patch(circle)

plt.xlim([-1.1,1.1])
plt.ylim([-1.1,1.1])

plt.scatter(pointsIn[:,0], pointsIn[:,1], color = 'green', s=0.05)
plt.scatter(pointsOut[:,0], pointsOut[:,1], color = 'red', s=0.05)
```

Out[9]:

```
<matplotlib.collections.PathCollection at 0x2269ebef5c0>
```




## Alternative: Loop method

In [10]:

```
i = 0
counter = 0

while i<points:
    if np.linalg.norm( 2*np.random.rand(2)-1 ) < 1:
        counter = counter + 1
    i = i+1
```

In [11]:

```
piapprox = 4 * counter / points
print(piapprox)
```

```
2.4
```

In [12]:

```
piapprox - np.pi
```

Out[12]:

```
-0.7415926535897932
```

## Time comparison

Array/List method

In [13]:

```
%%timeit

rand = 2*np.random.rand( 2*points ) - 1
randpoints = rand.reshape( points,2 )
normpoints = randpoints[:,0]**2 + randpoints[:,1]**2
pointsIn = randpoints[normpoints < 1]
piapprox = 4 * len(pointsIn) / points
```

```
14.2 µs ± 1.63 µs per loop (mean ± std. dev. of 7 runs, 100000 loops each)
```

Loop method

In [14]:

```
%%timeit

i = 0
counter = 0

while i<points:
    if np.linalg.norm( 2*np.random.rand(2)-1 ) < 1:
        counter = counter + 1
    i = i+1

piapprox = 4 * counter / points
```

```
124 µs ± 11.7 µs per loop (mean ± std. dev. of 7 runs, 10000 loops each)
```