In [185]:

```
import numpy as np
np.random.seed(312)

import matplotlib.pyplot as plt

from random import seed
from random import random
seed(312)
```

In [186]:

```
#define layers of neurons

class layer():

    def __init__(self, neurons, previous_nodes):
        self.weights = 2 * (np.random.random((previous_nodes, neurons)) - 0
        self.bias = np.zeros((1, neurons)) #set to zero for this example
```

In [187]:

```python
#define neural network itself

class NN():

    #initialize layers
    def __init__(self, layer1, layer2):
        self.layer1 = layer1
        self.layer2 = layer2
        self.loss = []

    #activation function is a sigmoid, output between 0 to 1,     S(x) = 1/
    def activation(self, x):
        return 1 / (1 + np.exp(-x))

    #backpropogation calculation, deriv of sig, how weight change impacts p
    def activation_derivative(self, x):
        return x * (1 - x)

    #relu function as alternative to sigmoid, R(x) = max(0,x)
    def relu(self,x):
        return max(0,x)

    #fitting network with iterations of forward prop, backprop, and adjustm
    def train(self, inputs, outputs, iterations):

        for iteration in range(0, iterations):

            #run calculation for current params
            output_layer_1, output_layer_2 = self.calculate(inputs)

            #figure out adjustments
            layer2_error = outputs - output_layer_2
            layer2_delta = layer2_error * self.activation_derivative(output_
            layer1_error = np.dot(layer2_delta,self.layer2.weights.T)
            layer1_delta = layer1_error * self.activation_derivative(output_
            layer1_adj = np.dot(inputs.T, layer1_delta)
            layer2_adj = np.dot(output_layer_1.T, layer2_delta)

            #adjust values
            self.layer1.weights += layer1_adj
            self.layer2.weights += layer2_adj

            #add iteration error to record
            self.loss.append(self.MSE(layer2_error))

    #calculate Mean Swuare Error Loss
    def MSE(self, error):
        total_error=0
        for i in error:
            total_error += i**2
        return total_error/len(error)

    #pass inputs through layers to get outputs
    def calculate(self, inputs):
        output_layer1 = self.activation(np.dot(inputs, self.layer1.weights)
        output_layer2 = self.activation(np.dot(output_layer1, self.layer2.w
        return output_layer1, output_layer2

```

```
    #print weights
    def show(self):
        print("")
        print("L1 ({} neurons): ".format(layer1.weights.shape[1]))
        print( self.layer1.weights)
        print("")
        print("L2 ({} neurons): ".format(layer2.weights.shape[1]))
        print(self.layer2.weights)
```

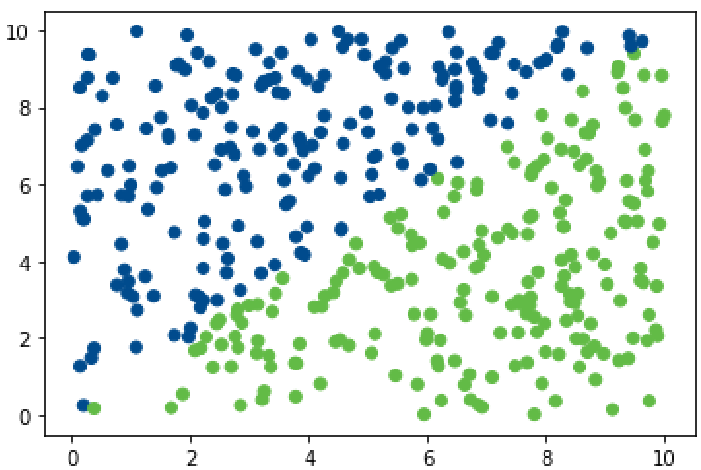
In [188]:

```
#data to train NN

A = []
B = []

for i in range(0,400):
    a = random()*10
    b = random()*10
    c = 1
    #if ((a-0)**2 + (b-0)**2) > (8**2-1):
    #     c=0
    if( b >= a ):
        c=0
    A.append([a,b])
    B.append([c])

X = np.array(A)
Y = np.array(B)
```

In [189]:

```
colors = ["#62BB46" if label ==1 else "#004B8D" for label in Y]
plt.scatter(X[:,0], X[:,1], color=colors)
```

Out[189]: <matplotlib.collections.PathCollection at 0x26ccdc1f188>



In [193]:

```
#initalize neural network, 4 neuron layer followed by 1 neuron layer
#important that second part of layer object is number of cols per input to
#we are feeding in data with 3 items, then second layer has 4 neurons in

layer1 = layer(2, X.shape[1])
layer2 = layer(1, layer1.weights.shape[1])
model = NN(layer1, layer2)

#train NN to fit data
model.train(X, Y, 5000)
```

```
C:\Users\Taylor_Vitunic\Miniconda3\envs\tensorflow\lib\site-packages\ipykernel
_launcher.py:13: RuntimeWarning: overflow encountered in exp
  del sys.path[0]
```

In [194]:

```
#test data
C = []
for i in range(0,20):
    for j in range(0,20):
        C.append([i/2,j/2])
C = np.array(C)
```

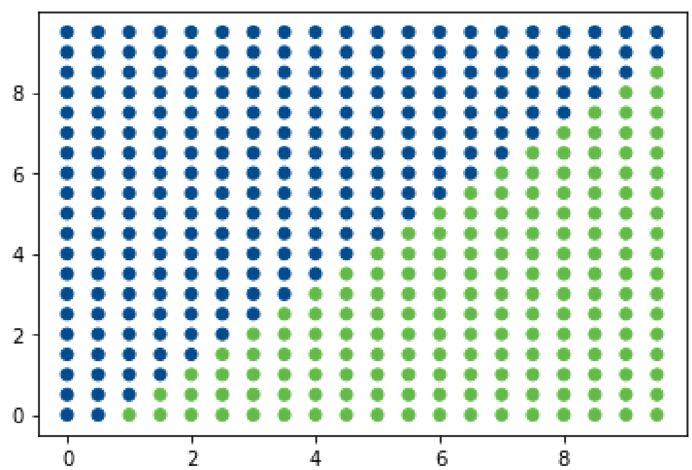
In [195]:

```
throwaway, output = model.calculate(C)
output = [1 if o >=.5 else 0 for o in output]
colors = ["#62BB46" if label ==1 else "#004B8D" for label in output]
plt.scatter(C[:,0], C[:,1], color=colors)
```

```
C:\Users\Taylor_Vitunic\Miniconda3\envs\tensorflow\lib\site-packages\ipykernel
_launcher.py:13: RuntimeWarning: overflow encountered in exp
  del sys.path[0]
```

Out[195]:

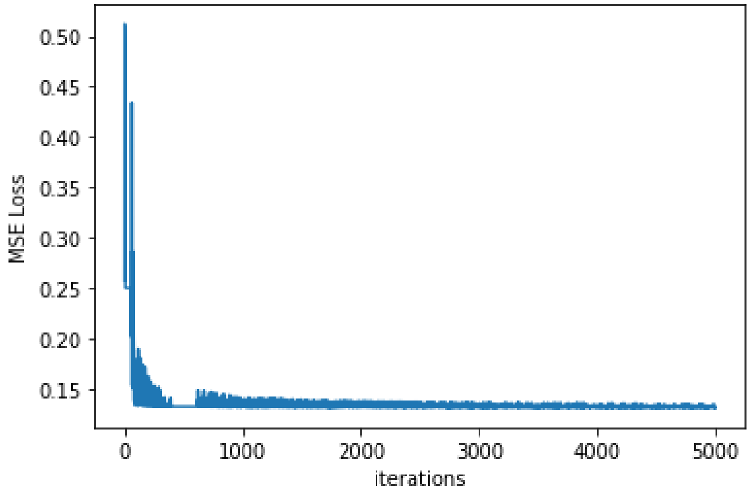
```
<matplotlib.collections.PathCollection at 0x26ccdd2fa08>
```



In [196]:

```
plt.plot(model.loss)
plt.xlabel("iterations")
plt.ylabel("MSE Loss")
```

Out[196]: Text(0, 0.5, 'MSE Loss')



In [ ]:

```
# here we are going to try a new approach
```

In [ ]:

```

```