In [ ]:

```
from PIL import Image, ImageDraw
import numpy as np
import random
import os.path
import pickle

#########################################################################
#                          Functions for you to complete
#########################################################################

def ComputeSSD(TODOPatch, TODOMask, textureIm, patchL):
        patch_rows, patch_cols, patch_bands = np.shape(TODOPatch)
        tex_rows, tex_cols, tex_bands = np.shape(textureIm)
        ssd_rows = tex_rows - 2 * patchL
        ssd_cols = tex_cols - 2 * patchL
        SSD = np.zeros((ssd_rows,ssd_cols))
        for r in range(ssd_rows):
                for c in range(ssd_cols):
                        # Compute sum square difference between textureIm
                        # for all pixels where TODOMask = 0, and store th
                        #
                        # ADD YOUR CODE HERE
                        #
                        textureIm.astype(np.float64)
                        TODOPatch.astype(np.float64)
                        red = np.sum(np.square(np.bitwise_and(textureIm[r
                        green = np.sum(np.square(np.bitwise_and(textureIm
                        blue = np.sum(np.square(np.bitwise_and(textureIm[
                        SSD[r,c] = red + green + blue
                        pass
                pass
        return SSD

def CopyPatch(imHole,TODOMask,textureIm,iPatchCenter,jPatchCenter,iMatchC
        patchSize = 2 * patchL + 1
        for i in range(patchSize):
                for j in range(patchSize):
                        # Copy the selected patch selectPatch into the im
                        # the hole imHole for each pixel where TODOMask =
                        # The patch is centred on iPatchCenter, jPatchCen
                        #
                        # ADD YOUR CODE HERE
                        #
                        imHole[iPatchCenter-patchL+i,jPatchCenter-patchL+
                        pass
                pass
        return imHole

#########################################################################
#                              Some helper functions
#########################################################################

def DrawBox(im,x1,y1,x2,y2):
        draw = ImageDraw.Draw(im)
        draw.line((x1,y1,x1,y2),fill="white",width=1)
        draw.line((x1,y1,x2,y1),fill="white",width=1)
        draw.line((x2,y2,x1,y2),fill="white",width=1)
        draw.line((x2,y2,x2,y1),fill="white",width=1)
        del draw
        return im

def Find_Edge(hole_mask):
```

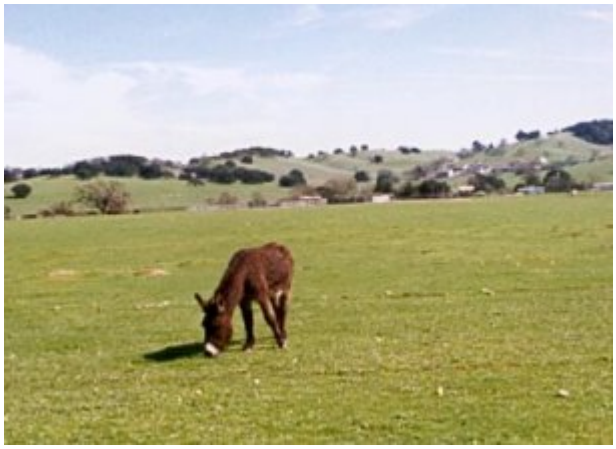
```
        [cols, rows] = np.shape(hole_mask)
        edge_mask = np.zeros(np.shape(hole_mask))
        for y in range(rows):
                for x in range(cols):
                        if (hole_mask[x,y] == 1):
                                if (hole_mask[x-1,y] == 0 or
                                                hole_mask[x+1,y] == 0 or
                                                hole_mask[x,y-1] == 0 or
                                                hole_mask[x,y+1] == 0):
                                        edge_mask[x,y] = 1
        return edge_mask

##########################################################################
#                           Main script starts here
##########################################################################
def main_script(fileName, patchL=10, randomPatchSD=1):
        #
        # Constants
        #

        # Change patchL to change the patch size used (patch size is 2 *p
        # patchL = 10
        patchSize = 2*patchL+1

        # Standard deviation for random patch selection
        # randomPatchSD = 1

        # Display results interactively
        showResults = False

        #
        # Read input image
        #

        im = Image.open(fileName).convert('RGB')
        im_array = np.asarray(im, dtype=np.uint8)
        imRows, imCols, imBands = np.shape(im_array)

        #
        # Define hole and texture regions.  This will use files fill_regi
        #   texture_region.pkl, if both exist, otherwise user has to sele
        if os.path.isfile('fill_region.pkl') and os.path.isfile('texture_
                fill_region_file = open('fill_region.pkl', 'rb')
                fillRegion = pickle.load( fill_region_file )
                fill_region_file.close()

                texture_region_file = open('texture_region.pkl', 'rb')
                textureRegion = pickle.load( texture_region_file )
                texture_region_file.close()
        else:
                # ask the user to define the regions
                print("Specify the fill and texture regions using polysel
                exit()

        #
        # Get coordinates for hole and texture regions
        #

        fill_indices = fillRegion.nonzero()
        nFill = len(fill_indices[0])                    # number of pixels to
        iFillMax = max(fill_indices[0])
        iFillMin = min(fill_indices[0])
        jFillMax = max(fill_indices[1])
        jFillMin = min(fill_indices[1])
```

```
        assert((iFillMin >= patchL) and
                        (iFillMax < imRows - patchL) and
                        (jFillMin >= patchL) and
                        (jFillMax < imCols - patchL)) , "Hole is too clos

        texture_indices = textureRegion.nonzero()
        iTextureMax = max(texture_indices[0])
        iTextureMin = min(texture_indices[0])
        jTextureMax = max(texture_indices[1])
        jTextureMin = min(texture_indices[1])
        textureIm   = im_array[iTextureMin:iTextureMax+1, jTextureMin:jTe
        texImRows, texImCols, texImBands = np.shape(textureIm)
        assert((texImRows > patchSize) and
                        (texImCols > patchSize)) , "Texture image is smal

        #
        # Initialize imHole for texture synthesis (i.e., set fill pixels
        #

        imHole = im_array.copy()
        imHole[fill_indices] = 0

        #
        # Is the user happy with fillRegion and textureIm?
        #
        if showResults == True:
                # original
                im.show()
                # convert to a PIL image, show fillRegion and draw a box
                im1 = Image.fromarray(imHole).convert('RGB')
                im1 = DrawBox(im1,jTextureMin,iTextureMin,jTextureMax,iTe
                im1.show()
                print("Are you happy with this choice of fillRegion and t
                Yes_or_No = False
                while not Yes_or_No:
                        answer = input("Yes or No: ")
                        if answer == "Yes" or answer == "No":
                                Yes_or_No = True
                assert answer == "Yes", "You must be happy. Please try ag

        #
        # Perform the hole filling
        #

        while (nFill > 0):
                print("Number of pixels remaining = " , nFill)

                # Set TODORegion to pixels on the boundary of the current
                TODORegion = Find_Edge(fillRegion)
                edge_pixels = TODORegion.nonzero()
                nTODO = len(edge_pixels[0])

                while(nTODO > 0):

                        # Pick a random pixel from the TODORegion
                        index = np.random.randint(0,nTODO)
                        iPatchCenter = edge_pixels[0][index]
                        jPatchCenter = edge_pixels[1][index]

                        # Define the coordinates for the TODOPatch
                        TODOPatch = imHole[iPatchCenter-patchL:iPatchCent
                        TODOMask = fillRegion[iPatchCenter-patchL:iPatchC

                        #
```

```
                        # Compute masked SSD of TODOPatch and textureIm
                        #
                        ssdIm = ComputeSSD(TODOPatch, TODOMask, textureIm

                        # Randomized selection of one of the best texture
                        ssdIm1 = np.sort(np.copy(ssdIm),axis=None)
                        ssdValue = ssdIm1[min(round(abs(random.gauss(0,ra
                        ssdIndex = np.nonzero(ssdIm==ssdValue)
                        iSelectCenter = ssdIndex[0][0]
                        jSelectCenter = ssdIndex[1][0]

                        # adjust i, j coordinates relative to textureIm
                        iSelectCenter = iSelectCenter + patchL
                        jSelectCenter = jSelectCenter + patchL
                        selectPatch = textureIm[iSelectCenter-patchL:iSel

                        #
                        # Copy patch into hole
                        #
                        imHole = CopyPatch(imHole,TODOMask,textureIm,iPat

                        # Update TODORegion and fillRegion by removing lo
                        TODORegion[iPatchCenter-patchL:iPatchCenter+patch
                        fillRegion[iPatchCenter-patchL:iPatchCenter+patch

                        edge_pixels = TODORegion.nonzero()
                        nTODO = len(edge_pixels[0])

                fill_indices = fillRegion.nonzero()
                nFill = len(fill_indices[0])

        #
        # Output results
        #
        if showResults == True:
                Image.fromarray(imHole).convert('RGB').show()
        Image.fromarray(imHole).convert('RGB').save(fileName.replace('.jp
```

In [ ]:
```
main_script("donkey.jpg")
display(Image.open("donkey.jpg"))
display(Image.open("donkey-results.jpg"))
```

```
Number of pixels remaining =  3473
Number of pixels remaining =  1043
Number of pixels remaining =  53
```



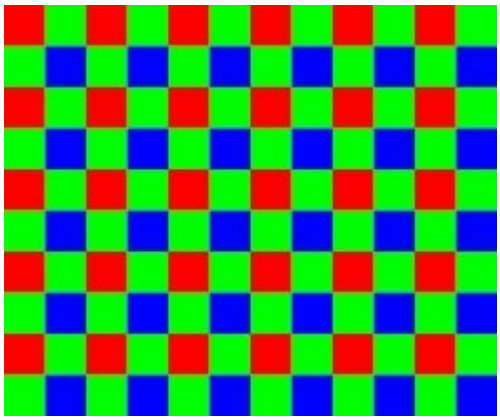
In [ ]:
```
main_script("dog.jpg")
display(Image.open("dog.jpg"))
display(Image.open("dog-results.jpg"))
# The grass in the background is quite difficult for the algorithm. There
```

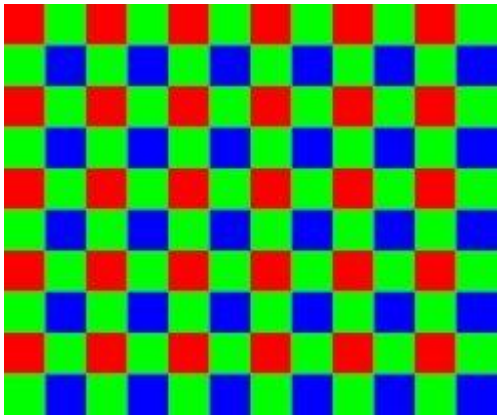
```
Number of pixels remaining =  4172
Number of pixels remaining =  1844
Number of pixels remaining =  322
```

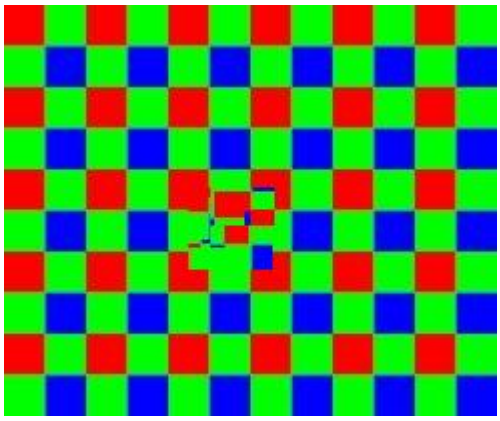
In [ ]:
```
main_script("ducks.jpg")
display(Image.open("ducks.jpg"))
display(Image.open("ducks-results.jpg"))
# In this case, the algorithm performs pretty well at replacing the duck.
```

```
Number of pixels remaining =  3072
Number of pixels remaining =  931
```

In [ ]:
```
# RandomPatchSD for greater randomization in the selection of the best te
# If this value is too small, for larger fills, there will be a lot of re
# If this value is too large, the texture patch will be very different fr
display(Image.open("bayer.jpg"))
main_script("bayer.jpg", randomPatchSD=0.1)
display(Image.open("bayer-results.jpg"))
main_script("bayer.jpg", randomPatchSD=10)
display(Image.open("bayer-results.jpg"))
# In this example, I run the algorithm on a blue square. With a low rando
# With a high randomPatchSD, the pool of possible texture patches is much
```







In [ ]:
```
# patchL is proportional to the size of the texture patch that the SSD an
# If this value is too small, the texture patch will be too small to capt
# If this value is too large, given that we do not have a Gaussian weight
display(Image.open("ducks.jpg"))
main_script("ducks.jpg", patchL=6)
display(Image.open("ducks-results.jpg"))
main_script("ducks.jpg", patchL=20)
display(Image.open("ducks-results.jpg"))
# In the first duck texture synthesis, you can see that the 'horizontal'
# In the second duck texture synthesis, while longer waves are captured,
# Therefore the more optimal value of 10 gets less noticeable artifacts.
```