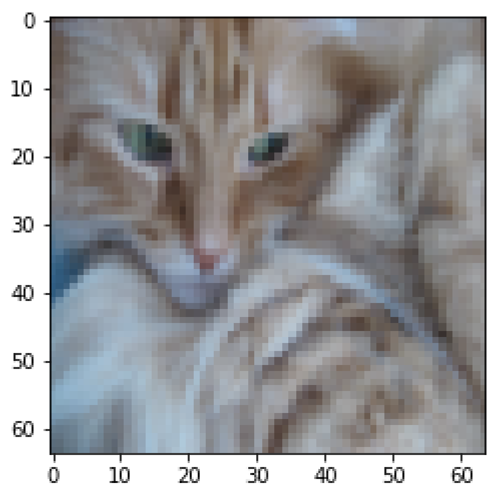
```
import numpy as np
import matplotlib.pyplot as plt
import h5py
import scipy
import PIL
from PIL import Image
from scipy import ndimage
from lr_utils import load_dataset


%matplotlib inline
```

```
help(PIL)
```

```
# Loading the data (cat/non-cat)
train_set_x_orig, train_set_y, test_set_x_orig, test_set_y, classes = load_dataset()
```

```
# Example of a picture
index = 2
plt.imshow(train_set_x_orig[index])
print ("y = " + str(train_set_y[:, index]) + ", it's a '" + classes[np.squeeze(train_set_y
```

```
y = [1], it's a 'cat' picture.
```



```
train_set_y
```

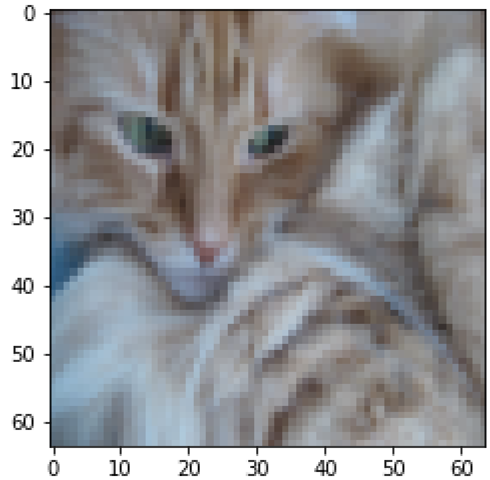
To undo cell deletion use Ctrl+M Z or the Undo option in the Edit menu

```
, 0, 0, 1, 0, 0,
, 0, 0, 1, 1, 0,
, 1, 0, 0, 0, 0,
0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0,
1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1,
1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0,
0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1,
0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1,
```

```
       0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1,
       0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0]])
```

```
plt.imshow(train_set_x_orig[2])
```

```
<matplotlib.image.AxesImage at 0x7f3aca880710>
```




```
classes
```

```
array([b'non-cat', b'cat'], dtype='|S7')
```

```
np.squeeze(train_set_y).shape
```

```
(209,)
```

```
train_set_y.shape
```

```
(1, 209)
```

so because train_set_y is an array of (1,209) np.squeez reduces one axis of train_set_y, therefore it changes to 209 which means there are 209 indeces and with class in front of it you basically go through all values (indeces) of train_set_y and print it's value if it's 0 its a non cat and if its 1 then its a cat

```
classes[np.squeeze(train_set_y[:, index])].decode("utf-8")
```

```
'cat'
```



```
bar = np.array([b'vvv',b'www'])
```

```
bar
```

```
array([b'vvv', b'www'], dtype='|S3')
```

```
testino = np.array([[0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0,
        0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0,
        0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0,
        0, 0, 1, 0, 0, 1, 0, 0, 0]])
```

```
testino
```

```
array([[0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0,
        0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0,
        0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0,
        0, 0, 1, 0, 0, 1, 0, 0, 0]])
```

```
bar[np.squeeze(testino[:,2])].decode("utf-8")
```

```
'www'
```

```
dt=np.dtype('|S3')
```

```
dt.itemsize
```

```
3
```

let's get the number of train and test samples

```
m_train = train_set_x_orig.shape[0]   #- m_train (number of training examples)
m_test = test_set_x_orig.shape[0]     #- m_test (number of test examples)
num_px = train_set_x_orig.shape[1]    #- num_px (= height = width of a training image)

print ("Number of training examples: m_train = " + str(m_train))
print ("Number of testing examples: m_test = " + str(m_test))
print ("Height/Width of each image: num_px = " + str(num_px))
print ("Each image is of size: (" + str(num_px) + ", " + str(num_px) + ", 3)")
print ("train_set_x shape: " + str(train_set_x_orig.shape))
print ("train_set_y shape: " + str(train_set_y.shape))
print ("test_set_x shape: " + str(test_set_x_orig.shape))
print ("test_set_y shape: " + str(test_set_y.shape))
```

To undo cell deletion use Ctrl+M Z or the Undo option in the Edit menu

```
Height/Width of each image: num_px = 64
Each image is of size: (64, 64, 3)
train_set_x shape: (209, 64, 64, 3)
train_set_y shape: (1, 209)
test_set_x shape: (50, 64, 64, 3)
test_set_y shape: (1, 50)
```

```
1 train_set_x_orig
```

```
        [[  8,   5,   0],
         [  9,   6,   1],
         [  9,   6,   1],
         ...,
         [  4,   5,   0],
         [  5,   4,   0],
         [  4,   5,   0]],

        [[  7,   5,   0],
         [  8,   5,   1],
         [  9,   6,   1],
         ...,
         [  4,   5,   0],
         [  4,   5,   0],
         [  4,   5,   0]],

        [[  7,   5,   0],
         [  8,   5,   0],
         [  9,   6,   1],
         ...,
         [  4,   5,   0],
         [  4,   5,   0],
         [  4,   5,   0]]],


       [[[  8,  28,  53],
         [ 14,  33,  58],
         [ 19,  35,  61],
         ...,
         [ 11,  16,  35],
         [ 10,  16,  35],
         [  9,  14,  32]],

        [[ 15,  31,  57],
         [ 15,  32,  58],
         [ 18,  34,  60],
         ...,
         [ 13,  17,  35],
         [ 13,  17,  35],
         [ 13,  16,  35]],

        [[ 20,  35,  61],
         [ 19,  33,  59],
         [ 20,  33,  59],
         ...,
```

To undo cell deletion use Ctrl+M Z or the Undo option in the Edit menu

```
        ...,

        [[  0,   0,   0],
         [  0,   0,   0],
         [  0,   0,   0],
```

```
        ...,
        [  0,   0,   0],
        [  0,   0,   0],
        [  0,   0,   0]],
```

```
train_set_x_orig[0][0]
```

```
array([[17, 31, 56],
       [22, 33, 59],
       [25, 35, 62],
       [25, 35, 62],
       [27, 36, 64],
       [28, 38, 67],
       [30, 41, 69],
       [31, 43, 73],
       [32, 47, 76],
       [34, 49, 79],
       [35, 50, 82],
       [36, 51, 82],
       [35, 50, 81],
       [34, 49, 79],
       [33, 48, 79],
       [33, 48, 79],
       [32, 47, 78],
       [31, 46, 76],
       [30, 44, 75],
       [29, 44, 75],
       [29, 44, 75],
       [27, 44, 74],
       [27, 42, 73],
       [25, 41, 71],
       [23, 40, 72],
       [21, 41, 73],
       [21, 42, 74],
       [21, 41, 74],
       [20, 40, 73],
       [20, 39, 72],
       [19, 39, 72],
       [18, 38, 71],
       [16, 38, 70],
       [14, 37, 69],
       [12, 37, 68],
       [11, 36, 67],
       [ 9, 36, 66],
       [ 7, 34, 64],
       [ 7, 35, 66],
       [ 4, 36, 69],
```



```
       [ 2, 34, 65],
       [ 1, 35, 67],
       [ 1, 34, 67],
       [ 1, 34, 66],
       [ 0, 32, 63],
       [ 1, 30, 61],
```

```
        [ 1, 30, 62],
        [ 2, 29, 59],
        [ 0, 29, 59],
        [ 1, 29, 59],
        [ 1, 28, 58],
        [ 1, 28, 57],
        [ 1, 28, 57],
        [ 1, 28, 57],
        [ 1, 28, 57],
        [ 1, 25, 55],
```

```
1 train_set_x_orig[0][1]
```

```
array([[25, 36, 62],
       [28, 38, 64],
       [30, 40, 67],
       [30, 39, 67],
       [31, 40, 68],
       [33, 41, 71],
       [34, 44, 73],
       [35, 45, 74],
       [35, 47, 75],
       [35, 48, 77],
       [36, 49, 78],
       [38, 51, 81],
       [37, 51, 82],
       [36, 49, 80],
       [36, 48, 79],
       [35, 48, 79],
       [34, 48, 79],
       [33, 46, 77],
       [32, 45, 76],
       [31, 45, 75],
       [30, 44, 74],
       [28, 43, 74],
       [27, 42, 73],
       [27, 41, 73],
       [25, 41, 72],
       [23, 41, 73],
       [23, 41, 73],
       [23, 40, 73],
       [23, 40, 71],
       [21, 40, 71],
       [21, 39, 70],
       [21, 39, 70],
       [19, 38, 70],
       [17, 38, 70],
```

To undo cell deletion use Ctrl+M Z or the Undo option in the Edit menu

```
       [10, 34, 65],
       [ 9, 34, 66],
       [ 6, 36, 69],
       [ 6, 37, 69],
       [ 4, 34, 66],
       [ 3, 33, 66],
```

```
            [ 3, 34, 67],
            [ 1, 34, 67],
            [ 1, 33, 65],
            [ 1, 32, 64],
            [ 1, 31, 63],
            [ 1, 30, 61],
            [ 0, 29, 59],
            [ 1, 29, 59],
            [ 1, 29, 59],
            [ 1, 28, 58],
            [ 1, 28, 57],
            [ 1, 27, 57],
            [ 1, 28, 58],
            [ 1, 28, 57],
            [ 1, 26, 56],
            [ 1, 27, 57],
```

```
train_set_x_orig.reshape(train_set_x_orig.shape[0],-1)
```

```
array([[ 17,  31,  56, ...,   0,   0,   0],
       [196, 192, 190, ...,  82,  80,  81],
       [ 82,  71,  68, ..., 138, 141, 142],
       ...,
       [143, 155, 165, ...,  85, 107, 149],
       [ 22,  24,  23, ...,   4,   5,   0],
       [  8,  28,  53, ...,   0,   0,   0]], dtype=uint8)
```

```
train_set_x_orig.reshape(train_set_x_orig.shape[0],-1).T[0]
```

```
array([ 17, 196,  82,   1,   9,  84,  56,  19,  63,  23, 188,   4, 154,
        17,  72, 245, 253, 217, 140,   2,   5,  17, 164, 156, 122,  15,
        78,  36,  14, 180,  39, 190, 233, 129, 137,  26,  23,  94,  63,
       113, 119,   1,  63, 255,  61,   0,  64,  51,  21,  57, 164, 152,
       106,  40,  15, 255,  31, 141,  52,  75,  81, 125,  99,  94,   2,
        86, 226,  76, 139,  43,  24,   7,  13, 103,  85, 110,  25,  61,
        34,  27, 176, 187,  26, 252,  96,  25,  34,  60, 123,  45,  99,
        49,  26, 154, 141,  62, 152, 194, 113,  57, 172,  70,  22, 142,
        37, 127, 172, 122, 110,  75, 165, 174,   5, 166, 144, 196,   2,
        64, 190, 170,  86, 106, 198,  70, 171,   9,  50,  84, 161,  23,
        79, 228, 104,   1,   5, 255, 142, 196, 135,  89,   0, 188, 255,
        17,  31, 169, 136,  79, 130, 150, 251,   7,  45, 159,  10, 135,
        32,  30, 140,  29,  29, 110,  99, 242, 158,  30, 240,  84,  10,
        93, 200, 190, 133,  74,  25,   3, 106, 133,  12, 105, 239,   1,
        62,  67,  29, 178,  68,  55, 201, 195, 144, 251, 130,  67,  10,
         0,  93, 101, 151,  29, 255,  43, 102,  93, 200,   9, 143,  22,
         8], dtype=uint8)
```

To undo cell deletion use Ctrl+M Z or the Undo option in the Edit menu

```
ape
```

```
(12288, 209)
```

```
train_set_x_orig.reshape(train_set_x_orig.shape[3],-1).T
```

```
array([[ 17,  72,   9],
```

```
       [ 31, 218,   9],
       [ 56, 159,  17],
       ...,
       [ 67,  13,   0],
       [212,  11,   0],
       [155,   8,   0]], dtype=uint8)
```

```
train_set_x_orig[208][0][1][2]
```

```
58
```

```
train_set_x_orig.shape
```

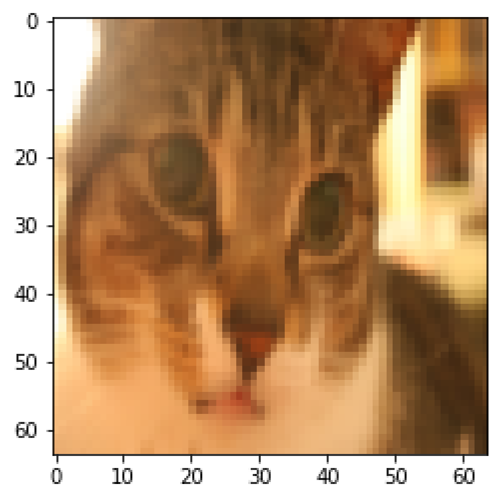
```
(209, 64, 64, 3)
```

```
train_set_x_orig[208][0][63]
```

```
array([ 9, 14, 32], dtype=uint8)
```

```
plt.imshow(train_set_x_orig[200])
```

```
<matplotlib.image.AxesImage at 0x7f3ac231a5c0>
```




```
train_set_x_orig[200][0][0:10]
```

```
array([[255, 255, 255],
       [255, 255, 255],
       [255, 255, 255],
       [255, 255, 255],
```

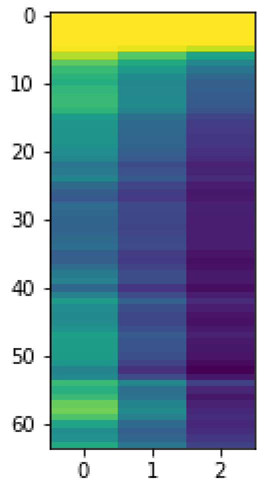


```
       [227, 192, 154],
       [195, 161, 126],
       [178, 144, 108],
       [169, 132,  96]], dtype=uint8)
```

```
plt.imshow(train_set_x_orig[200][0] , aspect=0.1)
```

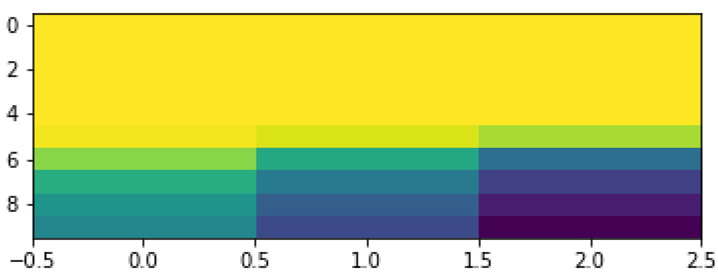
```
plt.imshow(train_set_x_orig[200][0] , aspect=0.1)
```

```
<matplotlib.image.AxesImage at 0x7f3ac2248748>
```




```
plt.imshow(train_set_x_orig[200][0][0:10] , aspect=0.1)
```

```
<matplotlib.image.AxesImage at 0x7f3ac21a7c88>
```




```

```

```
help(np.reshape)
```

```
# Reshape the training and test examples

### START CODE HERE ### (≈ 2 lines of code)

#train_set_x_flatten = None
#test_set_x_flatten = None

train_set_x_flatten = train_set_x_orig.reshape(train_set_x_orig.shape[0], -1).T
test_set_x_flatten = test_set_x_orig.reshape(test_set_x_orig.shape[0],-1).T
```

To undo cell deletion use Ctrl+M Z or the Undo option in the Edit menu

```
print ("train_set_x_flatten shape: " + str(train_set_x_flatten.shape))
print ("train_set_y shape: " + str(train_set_y.shape))
print ("test_set_x_flatten shape: " + str(test_set_x_flatten.shape))
print ("test_set_y shape: " + str(test_set_y.shape))
print ("sanity check after reshaping: " + str(train_set_x_flatten[0:5,0]))
```

```
train_set_x_flatten shape: (12288, 209)
train_set_y shape: (1, 209)
test_set_x_flatten shape: (12288, 50)
test_set_y shape: (1, 50)
sanity check after reshaping: [17 31 56 22 33]
```

```
train_set_x_flatten
```

```
array([[ 17, 196,  82, ..., 143,  22,   8],
       [ 31, 192,  71, ..., 155,  24,  28],
       [ 56, 190,  68, ..., 165,  23,  53],
       ...,
       [  0,  82, 138, ...,  85,   4,   0],
       [  0,  80, 141, ..., 107,   5,   0],
       [  0,  81, 142, ..., 149,   0,   0]], dtype=uint8)
```

```
train_set_x_flatten[0:2056,205]
```

```
array([ 9, 11, 13, ...,  6,  3,  5], dtype=uint8)
```

```
train_set_x_flatten[0:2056,0]
```

```
array([17, 31, 56, ..., 33, 61, 18], dtype=uint8)
```

```
train_set_x_flatten[1][208]
```

```
28
```

```
plt.imshow(train_set_x_flatten[1][208])
```

```
train_set_x = train_set_x_flatten/255.
test_set_x = test_set_x_flatten/255.
```

```
np.max(train_set_x_flatten)
```

```
255
```

```
train_set_x.shape
```

```
(12288, 209)
```

To undo cell deletion use Ctrl+M Z or the Undo option in the Edit menu ×

```
array([[ 17, 196,  82, ..., 143,  22,   8],
       [ 31, 192,  71, ..., 155,  24,  28],
       [ 56, 190,  68, ..., 165,  23,  53],
       ...,
       [  0,  82, 138, ...,  85,   4,   0],
```

```
           [  0,  80, 141, ..., 107,   5,   0],
           [  0,  81, 142, ..., 149,   0,   0]], dtype=uint8)
```

```
np.count_nonzero(train_set_x_flatten[2])
```

```
206
```

```
train_set_x_flatten.shape
```

```
(12288, 209)
```

```
train_set_x_flatten[[2001],[208]]
```

```
array([51], dtype=uint8)
```

```
train_set_x_flatten
```

```
array([[ 17, 196,  82, ..., 143,  22,   8],
       [ 31, 192,  71, ..., 155,  24,  28],
       [ 56, 190,  68, ..., 165,  23,  53],
       ...,
       [  0,  82, 138, ...,  85,   4,   0],
       [  0,  80, 141, ..., 107,   5,   0],
       [  0,  81, 142, ..., 149,   0,   0]], dtype=uint8)
```

```
train_set_y
```

```
array([[0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0,
        0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0,
        0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0,
        0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0,
        1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1,
        1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0,
        0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1,
        0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1,
        0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1,
        0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0]])
```

```
def sigmoid(z):
    """
    Compute the sigmoid of z

    z -- A scalar or numpy array of any size.

    Return:
    s -- sigmoid(z)
    """

```

To undo cell deletion use Ctrl+M Z or the Undo option in the Edit menu

```
    ### START CODE HERE ### (≈ 1 line of code)
    #s = None

    s = 1/(1+np.exp(-z))
    ### END CODE HERE ###

    return s
```

```
print("sigmoid([0,2])=" + str(sigmoid(np.array([0,2]))))
```

```
sigmoid([0,2])=[0.5        0.88079708]
```

```
sigmoid(2)
```

```
0.8807970779778823
```

```
# GRADED FUNCTION: initialize_with_zeros

def initialize_with_zeros(dim):
    """
    This function creates a vector of zeros of shape (dim, 1) for w and initializes b to 0

    Argument:
    dim -- size of the w vector we want (or number of parameters in this case)

    Returns:
    w -- initialized vector of shape (dim, 1)
    b -- initialized scalar (corresponds to the bias)
    """

    ### START CODE HERE ### (≈ 1 line of code)
    #w = None
    #b = None

    w = np.zeros(shape=(dim,1))
    b = 0

    ### END CODE HERE ###

    assert(w.shape == (dim, 1))
    assert(isinstance(b, float) or isinstance(b, int))

```

To undo cell deletion use Ctrl+M Z or the Undo option in the Edit menu ×

```
dim = 2
w, b = initialize_with_zeros(dim)
print('w is:', str(w))
print('b is:', str(b))
```

```
w is: [[0.]
 [0.]]
b is: 0
```

```
train_set_x_flatten.shape
```

```
(12288, 209)
```

```
(train_set_x_flatten.shape[0],2)
```

```
(12288, 2)
```

```
sdf=np.zeros(shape=(train_set_x.shape[0],2))
```

```
sdf.shape
```

```
(12288, 2)
```

```
np.zeros(shape=(train_set_x.shape[0],2))
```

```
array([[0., 0.],
       [0., 0.],
       [0., 0.],
       ...,
       [0., 0.],
       [0., 0.],
       [0., 0.]])
```

```

def propagate(w, b, X, Y):
    """
    Implement the cost function and its gradient for the propagation explained above

    Arguments:
    w -- weights, a numpy array of size (num_px * num_px * 3, 1)
    b -- bias, a scalar
    X -- data of size (num_px * num_px * 3, number of examples)
    Y -- true "label" vector (containing 0 if non-cat, 1 if cat) of size (1, number of exa

    Return:
    cost -- negative log-likelihood cost for logistic regression
                                                                  shape as w
                                                                  shape as b

    Tips:
    - Write your code step by step for the propagation. np.log(), np.dot()
    """

    m = X.shape[1]
```

```

    # FORWARD PROPAGATION (FROM X TO COST)
    ### START CODE HERE ### (≈ 2 lines of code)
    #A = None                                  # compute activation
    #cost = None                               # compute cost

   # A = sigmoid(w*train_set_x.shape + b)
    #cost = (- 1/m)(np.sum(Y*np.log(A)+(1 - Y)*np.log(1-A)))

    A = sigmoid(np.dot(w.T,X)+b)

    #cost = (- 1/m)(np.sum(np.dot(Y,np.log(A))+np.dot((1 - Y),np.log(1-A))))
    #cost = (-1/m) * np.sum(np.dot(Y,np.log(A)) + np.dot(1-Y, np.log(1-A)))
     #cost = (-1/m)*np.sum(np.dot(Y,np.log(A)) + np.dot((1-Y),(np.log(1 - A))))

    #simple multiply works :
    cost = (-1/m)*np.sum(Y*np.log(A) + (1-Y)*(np.log(1 - A)))

    # transpose of cost if get an error, also works
    #cost = (-1/m)*np.sum(np.dot(Y,np.log(A).T) + np.dot((1-Y),(np.log(1 - A)).T))

    ### END CODE HERE ###

    # BACKWARD PROPAGATION (TO FIND GRAD)
    ### START CODE HERE ### (≈ 2 lines of code)
   # dw = None
   # db = None


    #dw = 1/m*(X(A - Y).T)
    #dw = (1/m)*(X*(A-Y).T)                  why is it that you have to use dot product her
    dw = (1/m)*(np.dot(X,(A-Y).T))
    db = (1/m)*np.sum((A - Y))


    ### END CODE HERE ###

    assert(dw.shape == w.shape)
    assert(db.dtype == float)
    cost = np.squeeze(cost)
    assert(cost.shape == ())

    grads = {"dw": dw,
```

To undo cell deletion use Ctrl+M Z or the Undo option in the Edit menu ×

```
cost
```

```
0.6931471805599452
```

```
dim = 2
m = 3
b = 2
w = np.array([[1.],[2.]])
X = np.array([[1.,2.,-1.],[3.,4.,-3.2]])
Y = np.array([[1,0,1]])
A = sigmoid(np.dot(w.T,X)+2)
cost = (-1/m)*np.sum(Y*np.log(A) + (1-Y)*(np.log(1 - A)))
```

```
cost
```

```
5.801545319394553
```

```
X.shape[1]
```

```
3
```

```
# GRADED FUNCTION: optimize

def optimize(w, b, X, Y, num_iterations, learning_rate, print_cost = False):
    """
    This function optimizes w and b by running a gradient descent algorithm

    Arguments:
    w -- weights, a numpy array of size (num_px * num_px * 3, 1)
    b -- bias, a scalar
    X -- data of shape (num_px * num_px * 3, number of examples)
    Y -- true "label" vector (containing 0 if non-cat, 1 if cat), of shape (1, number of e
    num_iterations -- number of iterations of the optimization loop
    learning_rate -- learning rate of the gradient descent update rule
    print_cost -- True to print the loss every 100 steps

    Returns:
    params -- dictionary containing the weights w and bias b
    grads -- dictionary containing the gradients of the weights and bias with respect to t
    costs -- list of all the costs computed during the optimization, this will be used to

    Tips:
                                                                    through them:
                                                                    ent parameters. Use propagate()
                                                                    e for w and b.
    """

    costs = []

    for i in range(num_iterations):
```

To undo cell deletion use Ctrl+M Z or the Undo option in the Edit menu

```

        # Cost and gradient calculation (≈ 1-4 lines of code)
        ### START CODE HERE ###
       # grads, cost = None

        #grads, cost = {"dw": (1/m)*(np.dot(X,(A-Y).T)) , "db": 1/m*np.sum((A - Y)) } , (-

        grads, cost = propagate(w,b,X,Y)

        ### END CODE HERE ###

        # Retrieve derivatives from grads
        dw = grads["dw"]
        db = grads["db"]

        # update rule (≈ 2 lines of code)
        ### START CODE HERE ###
#   w = None
#   b = None
        w = w - learning_rate * dw
        b = b - learning_rate * db


        ### END CODE HERE ###

        # Record the costs
        if i % 100 == 0:
            costs.append(cost)

        # Print the cost every 100 training iterations
        if print_cost and i % 100 == 0:
            print ("Cost after iteration %i: %f" %(i, cost))

    params = {"w": w,
              "b": b}

    grads = {"dw": dw,
             "db": db}

    return params, grads, costs
```

```
params, grads, costs = optimize(w, b, X, Y, num_iterations= 100, learning_rate = 0.009, pr
```

To undo cell deletion use Ctrl+M Z or the Undo option in the Edit menu

```
print ("dw = " + str(grads["dw"]))
print ("db = " + str(grads["db"]))
```

```
w = [[0.19033591]
 [0.12259159]]
```

```
b = 1.9253598300845747
dw = [[0.67752042]
 [1.41625495]]
db = 0.21919450454067652
```

```
# GRADED FUNCTION: predict

def predict(w, b, X):
    '''
    Predict whether the label is 0 or 1 using learned logistic regression parameters (w, b

    Arguments:
    w -- weights, a numpy array of size (num_px * num_px * 3, 1)
    b -- bias, a scalar
    X -- data of size (num_px * num_px * 3, number of examples)

    Returns:
    Y_prediction -- a numpy array (vector) containing all predictions (0/1) for the exampl
    '''

    m = X.shape[1]
    Y_prediction = np.zeros((1,m))
    w = w.reshape(X.shape[0], 1)

    # Compute vector "A" predicting the probabilities of a cat being present in the pictur
    ### START CODE HERE ### (≈ 1 line of code)
    #A = None

    A = sigmoid(np.dot(w.T,X)+b)


    ### END CODE HERE ###

    for i in range(A.shape[1]):

        # Convert probabilities A[0,i] to actual predictions p[0,i]
        ### START CODE HERE ### (≈ 4 lines of code)
       # pass
    #    Y_prediction[0,i] = 0 if A[0,i] <=0.5 else 1 #np.dot(A[0],A[i])

       # y_prediction = []
        if A[0,i] <= 0.5:
            Y_prediction[0,i] = 0
```

To undo cell deletion use Ctrl+M Z or the Undo option in the Edit menu

```
      #   if A[i] <= 0.5:
      #      y_prediction = 0
      #  np.append(Y_prediction)



```

```

        ### END CODE HERE ###

    assert(Y_prediction.shape == (1, m))

    return Y_prediction
```

```
w = np.array([[0.1124579],[0.23106775]])
b = -0.3
X = np.array([[1.,-1.1,-3.2],[1.2,2.,0.1]])
print ("predictions = " + str(predict(w, b, X)))
```

```
predictions = [[1. 1. 0.]]
```

```
# GRADED FUNCTION: model

def model(X_train, Y_train, X_test, Y_test, num_iterations = 2000, learning_rate = 0.5, pr
    """
    Builds the logistic regression model by calling the function you've implemented previc

    Arguments:
    X_train -- training set represented by a numpy array of shape (num_px * num_px * 3, m_
    Y_train -- training labels represented by a numpy array (vector) of shape (1, m_train)
    X_test -- test set represented by a numpy array of shape (num_px * num_px * 3, m_test)
    Y_test -- test labels represented by a numpy array (vector) of shape (1, m_test)
    num_iterations -- hyperparameter representing the number of iterations to optimize the
    learning_rate -- hyperparameter representing the learning rate used in the update rule
    print_cost -- Set to true to print the cost every 100 iterations

    Returns:
    d -- dictionary containing information about the model.
    """

    ### START CODE HERE ###

    # initialize parameters with zeros (≈ 1 line of code)
    #w, b = None
    w, b = np.zeros(shape=((X_train.shape[0],1))) , 0

    # Gradient descent (≈ 1 line of code)
    # parameters, grads, costs = None
    parameters, grads, costs = optimize(w, b, X_train, Y_train, num_iterations, learning_r
```



```
rs"
    w = parameters["w"]
    b = parameters["b"]

    # Predict test/train set examples (≈ 2 lines of code)
    #Y_prediction_test = None
    #Y_prediction_train = None
```

```
    Y_prediction_test =  predict(w, b, X_test)
    Y_prediction_train = predict(w, b, X_train)

    ### END CODE HERE ###

    # Print train/test Errors
    print("train accuracy: {} %".format(100 - np.mean(np.abs(Y_prediction_train - Y_train)
    print("test accuracy: {} %".format(100 - np.mean(np.abs(Y_prediction_test - Y_test)) *


    d = {"costs": costs,
         "Y_prediction_test": Y_prediction_test,
         "Y_prediction_train" : Y_prediction_train,
         "w" : w,
         "b" : b,
         "learning_rate" : learning_rate,
         "num_iterations": num_iterations}

    return d
```

```
d = model(train_set_x, train_set_y, test_set_x, test_set_y, num_iterations = 2000, learnir
```

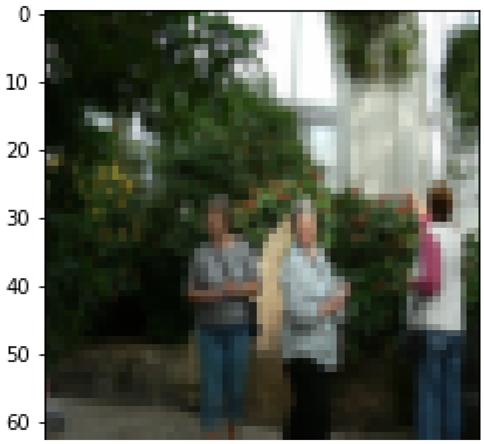
```
Cost after iteration 0: 0.693147
Cost after iteration 100: 0.584508
Cost after iteration 200: 0.466949
Cost after iteration 300: 0.376007
Cost after iteration 400: 0.331463
Cost after iteration 500: 0.303273
Cost after iteration 600: 0.279880
Cost after iteration 700: 0.260042
Cost after iteration 800: 0.242941
Cost after iteration 900: 0.228004
Cost after iteration 1000: 0.214820
Cost after iteration 1100: 0.203078
Cost after iteration 1200: 0.192544
Cost after iteration 1300: 0.183033
Cost after iteration 1400: 0.174399
Cost after iteration 1500: 0.166521
Cost after iteration 1600: 0.159305
Cost after iteration 1700: 0.152667
Cost after iteration 1800: 0.146542
Cost after iteration 1900: 0.140872
train accuracy: 99.04306220095694 %
test accuracy: 70.0 %
```

To undo cell deletion use Ctrl+M Z or the Undo option in the Edit menu

```
# Example of a picture that was wrongly classified.
index = 21
plt.imshow(test_set_x[:,index].reshape((num_px, num_px, 3)))
print("y = " + str(test_set_y[0,1]) + ", you predicted that it is a \"" + classes[d["Y_pre
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-407-2fcb190f7df1> in <module>()
      2 index = 21
      3 plt.imshow(test_set_x[:,index].reshape((num_px, num_px, 3)))
----> 4 print("y = " + str(test_set_y[0,1]) + ", you predicted that it is a \"" +
classes[d["Y_prediction_test"][0,index]].decode("utf-8") +  "\" picture.")

IndexError: only integers, slices (`:`), ellipsis (`...`), numpy.newaxis (`None`) and
integer or boolean arrays are valid indices
```

SEARCH STACK OVERFLOW



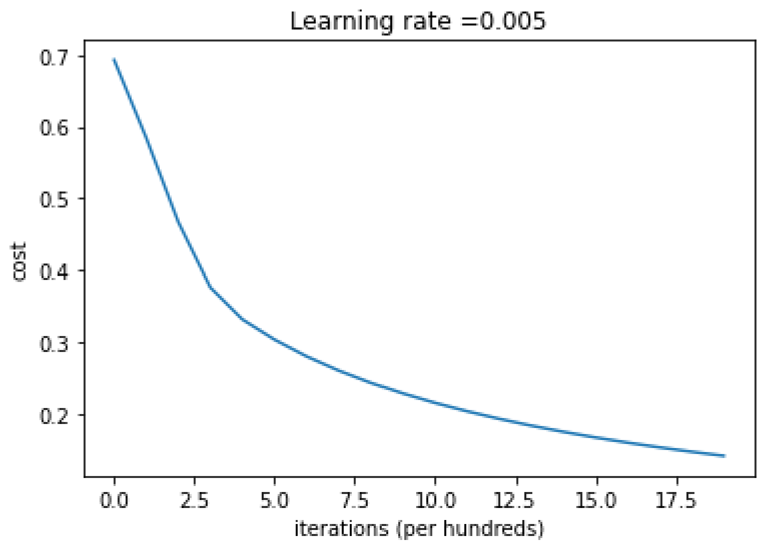
```
classes[d["Y_prediction_test"]][0,int(3)]
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-392-a9b0b409c198> in <module>()
----> 1 classes[d["Y_prediction_test"]][0,int(3)]

IndexError: arrays used as indices must be of integer (or boolean) type
```

SEARCH STACK OVERFLOW

```
# Plot learning curve (with costs)
costs = np.squeeze(d['costs'])
plt.plot(costs)
plt.ylabel('cost')
plt.xlabel('iterations (per hundreds)')
plt.title("Learning rate =" + str(d["learning_rate"]))
plt.show()
```

To undo cell deletion use Ctrl+M Z or the Undo option in the Edit menu



To undo cell deletion use Ctrl+M Z or the Undo option in the Edit menu