```
import numpy as np
```

```
from google.colab import files
uplaoded=files.upload()
```

Choose files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving sudoku_solution.npy to sudoku_solution.npy
Saving sudoku_game.npy to sudoku_game.npy
```

```
sudoku_game=np.load('sudoku_game.npy')
sudoku_game
```

```
array([[0, 0, 5, 0, 0, 9, 0, 0, 1],
       [0, 7, 0, 0, 6, 0, 0, 4, 3],
       [0, 0, 6, 0, 0, 2, 0, 8, 7],
       [1, 9, 0, 0, 0, 7, 4, 0, 0],
       [0, 5, 0, 0, 8, 3, 0, 0, 0],
       [6, 0, 0, 0, 0, 0, 1, 0, 5],
       [0, 0, 3, 5, 0, 8, 6, 9, 0],
       [0, 4, 2, 9, 1, 0, 3, 0, 0]])
```

+ Code + Text

Flattening & Reshaping of Data

```
#Faltten Sudoku_game
flatten_sudoku=sudoku_game.flatten()
flatten_sudoku
```

```
array([0, 0, 5, 0, 0, 9, 0, 0, 1, 0, 7, 0, 0, 6, 0, 0, 4, 3, 0, 0, 6, 0,
       0, 2, 0, 8, 7, 1, 9, 0, 0, 0, 7, 4, 0, 0, 0, 5, 0, 0, 8, 3, 0, 0,
       0, 6, 0, 0, 0, 0, 0, 1, 0, 5, 0, 0, 3, 5, 0, 8, 6, 9, 0, 0, 4, 2,
       9, 1, 0, 3, 0, 0])
```

```
sudoku_game
```

```
array([[0, 0, 5, 0, 0, 9, 0, 0, 1],
       [0, 7, 0, 0, 6, 0, 0, 4, 3],
       [0, 0, 6, 0, 0, 2, 0, 8, 7],
       [1, 9, 0, 0, 0, 7, 4, 0, 0],
       [0, 5, 0, 0, 8, 3, 0, 0, 0],
       [6, 0, 0, 0, 0, 0, 1, 0, 5],
       [0, 0, 3, 5, 0, 8, 6, 9, 0],
       [0, 4, 2, 9, 1, 0, 3, 0, 0]])
```

```
#Reshaping Concept
reshaped_sudoku=flatten_sudoku.reshape((9,8))
reshaped_sudoku
```

```
array([[0, 0, 5, 0, 0, 9, 0, 0],
       [1, 0, 7, 0, 0, 6, 0, 0],
       [4, 3, 0, 0, 6, 0, 0, 2],
       [0, 8, 7, 1, 9, 0, 0, 0],
       [7, 4, 0, 0, 0, 5, 0, 0],
       [8, 3, 0, 0, 0, 6, 0, 0],
       [0, 0, 0, 1, 0, 5, 0, 0],
       [3, 5, 0, 8, 6, 9, 0, 0],
       [4, 2, 9, 1, 0, 3, 0, 0]])
```

Numpy Data Types Numpy Vs Python Data Types Sample Python Data Types

1. int
2. Float

Sample Numpy Datatypes

1. np.int64
2. np.int32
3. np.float64
4. np.float32

The .dtype Attribute

```
np.array([1.32,5.78,175.175]).dtype
```

```
    dtype('float64')
```

String Data

```
np.array(['Introduction','to','Numpy']).dtype

    dtype('<U12')
```

Dtype as an arguments

```
float_32_array=np.array([1.32,5.78,175.175],dtype=np.float32)
float_32_array
float_32_array.dtype

    dtype('float32')
```

Type Conversion

```
boolean_arr=np.array([[True,False],[False,False]],dtype=np.bool_)
boolean_arr.astype(np.int32)
#boolean_arr

    <ipython-input-5-10f12a7faf92>:1: DeprecationWarning: `np.bool` is a deprecated alias for the builtin `bool`. To silence
    Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/release/1.20.0-notes.html#deprecation
      boolean_arr=np.array([[True,False],[False,False]],dtype=np.bool)
    array([[1, 0],
           [0, 0]], dtype=int32)
```

Type Coercion

```
np.array([True,"BillGates",42,42.42])

    array(['True', 'BillGates', '42', '42.42'], dtype='<U32')
```

Type Coercion Hierarchy: Adding a Float to an array of integers will change all integers into float?

```
np.array([0,42,42.42]).dtype

    dtype('float64')
```

Adding an integer to an array of booleans will change all booleans in to int

```
np.array([True,False,99]).dtype

    dtype('int64')
```

```
# Create an Array of Zeros with Three Rows and Two Cols
zero_arr=np.zeros((3,2))
zero_arr

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

Assignment: np.array([34.62,70.13,True,"TOM"]).astype(np.int64) ==>>?

```
sudoku_game.dtype # Trying to Find datatype of Sudoku Game - Thridparty Numpy Array

    dtype('int64')
```

```
sudoku_game

    array([[0, 0, 5, 0, 0, 9, 0, 0, 1],
           [0, 7, 0, 0, 6, 0, 0, 4, 3],
           [0, 0, 6, 0, 0, 2, 0, 8, 7],
           [1, 9, 0, 0, 0, 7, 4, 0, 0],
           [0, 5, 0, 0, 8, 3, 0, 0, 0],
           [6, 0, 0, 0, 0, 0, 1, 0, 5],
           [0, 0, 3, 5, 0, 8, 6, 9, 0],
           [0, 4, 2, 9, 1, 0, 3, 0, 0]])
```

Indexing 1D Array

```
arr_1=np.array([2,4,6,8,10])
arr_1
```

```
    array([ 2,  4,  6,  8, 10])
```

```
arr_1[3]
```

```
    8
```

```
sudoku_game
```

```
    array([[0, 0, 5, 0, 0, 9, 0, 0, 1],
           [0, 7, 0, 0, 6, 0, 0, 4, 3],
           [0, 0, 6, 0, 0, 2, 0, 8, 7],
           [1, 9, 0, 0, 0, 7, 4, 0, 0],
           [0, 5, 0, 0, 8, 3, 0, 0, 0],
           [6, 0, 0, 0, 0, 0, 1, 0, 5],
           [0, 0, 3, 5, 0, 8, 6, 9, 0],
           [0, 4, 2, 9, 1, 0, 3, 0, 0]])
```

```
sudoku_game[2,4]
```

```
    0
```

```
arr_1
```

```
    array([ 2,  4,  6,  8, 10])
```

```
arr_1[3]
```

```
    8
```

```
arr_1[:3]
```

```
    array([2, 4, 6])
```

```
arr_1[3:]
```

```
    array([ 8, 10])
```

```
sudoku_game[:,3]
```

```
    array([0, 0, 0, 0, 0, 0, 5, 9])
```

```
#Slicing 1D Array
arr_1[2:4]
arr_1
```

```
    array([ 2,  4,  6,  8, 10])
```

```
arr_1[2:4]
```

```
    array([6, 8])
```

```
#Slicing 2D Array
sudoku_game[3:6,3:6]
```

```
    array([[0, 0, 7],
           [0, 8, 3],
           [0, 0, 0]])
```

```
sudoku_game[3:6:2, 3:6:2] # What Expected?
```

```
    array([[0, 7],
           [0, 0]])
```

```
sudoku_game
```

```
    array([[0, 0, 5, 0, 0, 9, 0, 0, 1],
           [0, 7, 0, 0, 6, 0, 0, 4, 3],
           [0, 0, 6, 0, 0, 2, 0, 8, 7],
           [1, 9, 0, 0, 0, 7, 4, 0, 0],
           [0, 5, 0, 0, 8, 3, 0, 0, 0],
           [6, 0, 0, 0, 0, 0, 1, 0, 5],
           [0, 0, 3, 5, 0, 8, 6, 9, 0],
           [0, 4, 2, 9, 1, 0, 3, 0, 0]])
```

```
#Sortring an array?
np.sort(sudoku_game)
```

```
    array([[0, 0, 0, 0, 0, 0, 1, 5, 9],
           [0, 0, 0, 0, 0, 3, 4, 6, 7],
           [0, 0, 0, 0, 0, 2, 6, 7, 8],
           [0, 0, 0, 0, 0, 1, 4, 7, 9],
           [0, 0, 0, 0, 0, 0, 3, 5, 8],
           [0, 0, 0, 0, 0, 0, 1, 5, 6],
           [0, 0, 0, 0, 3, 5, 6, 8, 9],
           [0, 0, 0, 0, 1, 2, 3, 4, 9]])
```

Keep in Mind in a Matrix Rows Considered as Axis=1 Cols Considers as Axis=0

```
np.sort(sudoku_game,axis=1) # You can make axis=0 too
```

```
    array([[0, 0, 0, 0, 0, 0, 1, 5, 9],
           [0, 0, 0, 0, 0, 3, 4, 6, 7],
           [0, 0, 0, 0, 0, 2, 6, 7, 8],
           [0, 0, 0, 0, 0, 1, 4, 7, 9],
           [0, 0, 0, 0, 0, 0, 3, 5, 8],
           [0, 0, 0, 0, 0, 0, 1, 5, 6],
           [0, 0, 0, 0, 3, 5, 6, 8, 9],
           [0, 0, 0, 0, 1, 2, 3, 4, 9]])
```

```
from google.colab import files
uplaoded=files.upload()
```

Choose files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
    Saving tree_census.npy to tree_census.npy
```

Imagine You area researcher working with data from New York City Tree census. Each Row of the tree_census . 2D Array List - Capstone Project

Select All rows of block id data from the second columns

```
tree_census=np.load('tree_census.npy')
tree_census
```

```
    array([[     3, 501451,     24,      0],
           [     4, 501451,     20,      0],
           [     7, 501911,      3,      0],
           ...,
           [  1198, 227387,     11,      0],
           [  1199, 227387,     11,      0],
           [  1210, 227386,      6,      0]])
```

```
block_ids=tree_census[:,1]
block_ids
```

```
    array([501451, 501451, 501911, 501911, 501911, 501911, 501911, 501911,
           501911, 501911, 501911, 501911, 501909, 501909, 501909, 501909,
           501909, 501909, 501909, 501967, 501967, 501967, 501134, 501134,
           501134, 501134, 501134, 501134, 501134, 501134, 501966, 501966,
           501966, 501966, 501966, 501966, 500846, 500846, 500846, 500846,
           500846, 500846, 500836, 500836, 500836, 500836, 500836, 500836,
           500836, 500836, 500836, 500836, 500836, 501882, 501882, 501882,
           501882, 501882, 501882, 501882, 501882, 501882, 501882, 501113,
           501113, 501113, 501113, 501113, 500917, 500917, 500917, 501323,
           501323, 501323, 501323, 501323, 501323, 501323, 501451, 501451,
           516296, 516296, 516296, 516296, 516296, 516296, 516296, 516296,
           516296, 516296, 516296, 516296, 516296, 516296, 503178, 503178,
           503178, 503178, 503178, 503178, 501451, 501451, 501451, 501451,
           501451, 501451, 501451, 501451, 501451, 501874, 501874, 501874,
           501874, 501874, 501874, 501874, 501874, 501874, 501874, 501874,
           501874, 501874, 501874, 501874, 501874, 501874, 501874, 501874,
           501874, 501874, 501874, 501874, 501874, 501874, 501874, 501874,
           501874, 501874, 501874, 501874, 501874, 501874, 501874, 501874,
           501874, 501874, 501874, 501874, 501942, 501942, 501942, 501942,
           314009, 314009, 314009, 314009, 314009, 314009, 314009, 314009,
           314009, 314009, 314009, 314009, 314009, 314009, 314009, 314009,
           314009, 314009, 314009, 314009, 314009, 314009, 314009, 501867,
           501867, 501841, 501841, 501841, 501892, 501892, 501892, 501892,
           501892, 501892, 501892, 501892, 501892, 501892, 501892, 501892,
           501892, 501892, 501892, 501892, 501892, 501892, 501897, 501897,
           501897, 501897, 501897, 501897, 501897, 501897, 501897, 501897,
           501897, 501897, 501897, 501897, 501897, 501897, 501897, 501897,
           501897, 501897, 501897, 501893, 501893, 501893, 501893, 501893,
           501893, 501893, 501893, 501893, 501893, 501893, 501893, 107255,
           107255, 107255, 107255, 413051, 413051, 413051, 413051, 413051,
           413051, 413051, 413051, 413051, 413051, 413051, 413051, 413051,
           413051, 413051, 413051, 413051, 413051, 413051, 413051, 413051,
           413051, 413051, 413051, 413051, 413051, 413052, 413052, 413052,
           413052, 413052, 413052, 413052, 413052, 107288, 107288, 107288,
           413086, 413086, 413086, 413086, 413086, 413086, 413086, 413086,
           413086, 413086, 207803, 207803, 207803, 207803, 207803, 207803,
           207803, 207803, 207803, 107278, 107278, 107278, 107188, 107188,
           107188, 107188, 107188, 207088, 207088, 207088, 207088, 107256,
           107256, 107256, 207945, 207945, 207945, 207945, 207945, 207112,
           207112, 207112, 507568, 507568, 507568, 107277, 107277, 107277,
           107277, 107277, 107277, 107277, 107277, 107285, 107285, 107285,
           107285, 207802, 207802, 207802, 207041, 207041, 207041, 207041,
           207041, 516271, 516271, 516271, 516271, 516271, 516271, 207199,
           207199, 207199, 207199, 207199, 207199, 207199, 207199, 207199,
           207199, 507598, 507598, 207201, 207201, 507613, 507613, 507613,
           507613, 507599, 507599, 507599, 507599, 507599, 507599, 507599,
           507599, 507599, 507599, 207200, 207200, 207200, 207200, 207200,
           207200, 207200, 207200, 207200, 207200, 207200, 207200, 207200,
           507614, 507614, 507539, 507539, 507539, 507539, 507539, 507539,
           507615, 507615, 507615, 507615, 507615, 507615, 343368, 343368,
           349684, 349684, 349684, 349687, 349687, 349687, 349687, 342452,
           342452, 342452, 342452, 342453, 342453, 342453, 342453, 344023,
           344023, 344023, 344023, 344023, 344023, 344023, 342454, 342454,
           342454, 342454, 342454, 342454, 349688, 349688, 349688, 349688,
           349688, 349688, 349688, 349688, 349688, 349688, 349688, 349688,
           349688, 349688, 349688, 349688, 342455, 342455, 342455, 342455,
           342455, 342456, 342456, 342457, 342457, 342457, 342457, 342457,
```

```
# Print the First Five block_ids
block_ids[:5]
```

```
    array([501451, 501451, 501911, 501911, 501911])
```

```
#Steeping into 2D – Now Assume that your research requires you to take an admittedly unrespresentaive sample of trunk diamet
```

```
# Create an array of the first 100 trunk diameters from tree census
hundred_dia=tree_census[:100,2]
hundred_dia
```

```
    array([24, 20,  3,  3,  4,  4,  4,  4,  4,  3,  3,  4,  2,  2,  3,  4,  4,
            4,  0, 14,  3,  4,  7,  8,  7,  8,  7,  5,  6,  5,  5, 17,  0, 19,
           21, 18,  4,  5,  3,  4,  3,  4, 13, 13, 13,  5,  4,  4,  4, 11,  5,
            4,  5,  8, 51,  7,  4, 15,  3,  8,  6,  6,  3,  4,  3,  2,  3,  3,
            6,  5,  5,  5,  5,  9,  4,  4,  7,  7,  6,  5,  4,  4,  5,  5,  5,
            7,  3,  5,  3,  3,  6,  6,  8,  7,  4,  5,  4,  4,  4,  4])
```

Extract trunk diameter information from smallest to largest

```
sorted_trunk_dia=np.sort(tree_census[:,2])
sorted_trunk_dia
```

```
    array([ 0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
            0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
            0,  0,  1,  1,  1,  1,  1,  1,  2,  2,  2,  2,  2,  2,  2,  2,  2,
            2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
            2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
            2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
            2,  2,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,
            3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,
            3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,
            3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,
            3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,
            3,  4,  4,  4,  4,  4,  4,  4,  4,  4,  4,  4,  4,  4,  4,  4,  4,
            4,  4,  4,  4,  4,  4,  4,  4,  4,  4,  4,  4,  4,  4,  4,  4,  4,
            4,  4,  4,  4,  4,  4,  4,  4,  4,  4,  4,  4,  4,  4,  4,  4,  4,
            4,  4,  4,  4,  4,  4,  4,  4,  4,  4,  4,  4,  4,  4,  4,  4,  4,
            4,  4,  4,  4,  4,  4,  4,  4,  4,  4,  4,  4,  4,  4,  4,  4,  4,
            4,  4,  4,  4,  4,  4,  4,  4,  4,  4,  4,  4,  4,  4,  4,  4,  4,
            4,  4,  4,  4,  4,  4,  4,  5,  5,  5,  5,  5,  5,  5,  5,  5,  5,
            5,  5,  5,  5,  5,  5,  5,  5,  5,  5,  5,  5,  5,  5,  5,  5,  5,
            5,  5,  5,  5,  5,  5,  5,  5,  5,  5,  5,  5,  5,  5,  5,  5,  5,
            5,  5,  5,  5,  5,  5,  5,  5,  5,  5,  5,  5,  5,  5,  5,  5,  5,
            5,  5,  5,  5,  5,  5,  5,  5,  5,  5,  5,  5,  5,  6,  6,  6,  6,
            6,  6,  6,  6,  6,  6,  6,  6,  6,  6,  6,  6,  6,  6,  6,  6,  6,
            6,  6,  6,  6,  6,  6,  6,  6,  6,  6,  6,  6,  6,  6,  6,  6,  6,
            6,  6,  6,  6,  6,  6,  6,  6,  6,  6,  6,  6,  6,  6,  6,  6,  6,
            7,  7,  7,  7,  7,  7,  7,  7,  7,  7,  7,  7,  7,  7,  7,  7,  7,
            7,  7,  7,  7,  7,  7,  7,  7,  7,  7,  7,  7,  7,  7,  7,  7,  7,
            7,  7,  7,  7,  7,  7,  7,  7,  7,  7,  7,  7,  7,  7,  7,  7,  7,
            8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,
            8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,
            8,  8,  9,  9,  9,  9,  9,  9,  9,  9,  9,  9,  9,  9,  9,  9,  9,
            9,  9,  9,  9,  9,  9,  9,  9,  9,  9,  9,  9,  9,  9,  9,  9,  9,
            9,  9,  9,  9,  9,  9,  9,  9,  9, 10, 10, 10, 10, 10, 10, 10, 10,
           10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10,
           10, 10, 10, 10, 10, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11,
           11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11,
           11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11,
           11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 12, 12, 12, 12, 12, 12, 12,
           12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12,
           12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12,
           12, 12, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13,
           13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13,
           13, 13, 13, 13, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14,
           14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14,
           14, 14, 14, 14, 14, 14, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15,
           15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15,
           15, 15, 15, 15, 15, 15, 15, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16,
           16, 16, 16, 16, 16, 16, 16, 16, 16, 17, 17, 17, 17, 17, 17, 17, 17,
           17, 17, 17, 17, 17, 17, 17, 17, 17, 17, 17, 17, 17, 17, 17, 17, 17,
           17, 17, 17, 17, 18, 18, 18, 18, 18, 18, 18, 18, 18, 18, 18, 18, 18,
           18, 18, 18, 18, 18, 18, 18, 18, 18, 18, 18, 18, 18, 18, 19, 19, 19,
           19, 19, 19, 19, 19, 19, 19, 19, 19, 19, 19, 19, 19, 19, 19, 19, 19,
           19, 19, 19, 19, 19, 19, 19, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20,
           20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 21, 21, 21, 21,
           21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 22, 22,
           22, 22, 22, 22, 22, 22, 22, 22, 22, 22, 22, 22, 22, 22, 22, 23, 23,
           23, 23, 23, 23, 23, 23, 23, 23, 23, 24, 24, 24, 24, 24, 24, 24, 24,
           25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 26, 26, 26, 26, 26, 26,
```

Concept of Filtering

1. Masks and Fancy Indexing
2. np.where()

```
#Boolean Masking
one_to_five=np.arange(1,6)
one_to_five
```

```
    array([1, 2, 3, 4, 5])
```

```
mask=one_to_five % 2==0
mask
```

```
    array([False,  True, False,  True, False])
```

```
one_to_five[mask]
```

```
    array([2, 4])
```

```
# Implement the same with 2D Ones 2D Fancy Indexing
classroom_ids_and_sizes=np.array([[1,22],[2,21],[3,27],[4,26]])
classroom_ids_and_sizes
```

```
    array([[ 1, 22],
           [ 2, 21],
           [ 3, 27],
           [ 4, 26]])
```

```
classroom_ids_and_sizes[:,0][classroom_ids_and_sizes[:,1]%2==0]
```

```
    array([1, 4])
```

```
#Filtering with np.where?
np.where(classroom_ids_and_sizes[:,1]%2==0)
```

```
    (array([0, 3]),)
```