In [8]:
```
import numpy as np
```

In [9]:
```
# Task 1: Create 3D array having two rows and two columns and 10
# parallel
a = np.ones((10,2,2))
print(a)
```

```
[[[1. 1.]
  [1. 1.]]

 [[1. 1.]
  [1. 1.]]

 [[1. 1.]
  [1. 1.]]

 [[1. 1.]
  [1. 1.]]

 [[1. 1.]
  [1. 1.]]

 [[1. 1.]
  [1. 1.]]

 [[1. 1.]
  [1. 1.]]

 [[1. 1.]
  [1. 1.]]

 [[1. 1.]
  [1. 1.]]

 [[1. 1.]
  [1. 1.]]]
```

In [10]:
```
# Task 2: Convert a 4D Numpy array having 24 elements into a 2D array
# having square of each element.
fourDArray = np.array([[
              [[1,2],[3,4]],
              [[5,6],[7,8]],
              [[9,10],[11,12]]
              ],
              [
              [[13,14],[15,16]],
              [[17,18],[19,20]],
              [[21,22],[23,24]]
              ]])
twoDArray = fourDArray.reshape(6,4)**2
print(twoDArray)
```

```
[[  1   4   9  16]
 [ 25  36  49  64]
 [ 81 100 121 144]
 [169 196 225 256]
 [289 324 361 400]
 [441 484 529 576]]
```

In [84]:
```
# Task 3: Make a list of 1000 elements between 0 and 1. Calculate square
# of each element and print time taken for execution.
# Repeat it for Numpy and compare time. Increase elements up to 10000 and 1000000
# and see results.
#first Part
import time as t
start_time = t.time()
value = 0
thousandEleArray = list()
for i in range(0,1000):
    value+=0.001
    formatted_string = "{:.4f}".format(value)
    float_value = float(formatted_string)
    thousandEleArray.append(float_value)
%timeit thousandSquEleArray = [i**2 for i in thousandEleArray]

print("For 1000 numpy array",t.time() - start_time," milliseconds has been taken")
```

```
407 µs ± 12 µs per loop (mean ± std. dev. of 7 runs, 1,000 loops each)
For 1000 numpy array 3.3077075481414795  milliseconds has been taken
```

In [83]:
```
#second part

new_start_time = t.time()
print("using numpyArray")
x = np.arange(0,1,0.001)
%timeit y = x**2
print("For 1000 numpy array",t.time() - new_start_time," milliseconds has been taken")
x = np.arange(0,1,0.0001)
%timeit y = x**2
print("For 10000 numpy array",t.time() - new_start_time," milliseconds has been taken"
x = np.arange(0,1,0.00001)
%timeit y = x**2
print("For 10000 numpy array",t.time() - new_start_time," milliseconds has been taken"
```

```
using numpyArray
2.5 µs ± 20.7 ns per loop (mean ± std. dev. of 7 runs, 100,000 loops each)
For 1000 numpy array 2.125077724456787  milliseconds has been taken
8.98 µs ± 194 ns per loop (mean ± std. dev. of 7 runs, 100,000 loops each)
For 10000 numpy array 9.404545783996582  milliseconds has been taken
88.7 µs ± 1.7 µs per loop (mean ± std. dev. of 7 runs, 10,000 loops each)
For 10000 numpy array 16.604185104370117  milliseconds has been taken
```

In [14]:
```
# Task 4: Make a Numpy array having 5 rows and 5 columns using ones( )function
# After that convert it into the following shape:
# [[1., 1., 1., 1., 1.],
# [1., 0., 0., 0., 1.],
# [1., 0., 0., 0., 1.],
# [1., 0., 0., 0., 1.],
# [1., 1., 1., 1., 1.]]
ones = np.ones((5,5))
zeros = np.zeros((3,3))
ones[1:4 , 1:4] = zeros
print(ones)
```

```
[[1. 1. 1. 1. 1.]
 [1. 0. 0. 0. 1.]
 [1. 0. 0. 0. 1.]
 [1. 0. 0. 0. 1.]
 [1. 1. 1. 1. 1.]]
```