Elective Course

Course Code: CS4103

Autumn 2025-26

Lecture #06

Artificial Intelligence for Data Science

Week-2:

Python Primer for AI (Part-II)

Course Instructor:

Dr. Monidipa Das

Assistant Professor

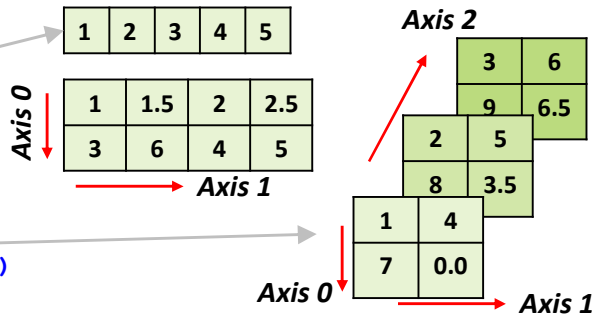
Department of Computational and Data Sciences

Indian Institute of Science Education and Research Kolkata, India 741246

A Few Python Libraries for AI

NumPy: Creating Array

```
import numpy as np

a = np.array([1,2,3,4,5])

b = np.array([(1,1.5,2,2.5), (3,6,4,5)])

c = np.array([[(1,2,3), (4,5,6)],
[(7,8,9),(0.0,3.5,6.5)]],dtype='float32')
```




```
print("Array 'a' has Type:",a.dtype,"; Dimension:",a.ndim,"; Shape:",a.shape, "; Size:",a.size )
print("Array 'b' has Type:",b.dtype,"; Dimension:",b.ndim,"; Shape:",b.shape,"; Size:",b.size)
print("Array 'c' has Type:",c.dtype,"; Dimension:",c.ndim,"; Shape:",c.shape,"; Size:",c.size)
print("Length of first dimension for 'a', 'b', 'c':",len(a),len(b),len(c))
```

```
Array 'a' has Type: int32 ; Dimension: 1 ; Shape: (5,) ; Size: 5
Array 'b' has Type: float64 ; Dimension: 2 ; Shape: (2, 4) ; Size: 8
Array 'c' has Type: float32 ; Dimension: 3 ; Shape: (2, 2, 3) ; Size: 12
Length of first dimension for 'a', 'b', 'c': 5 2 2
```

NumPy: Creating Array


```
#Changing array types
b=b.astype(int)
print(b)

#Create an array of zeros
a1=np.zeros((5))
print("a1=\n",a1)
b1=np.zeros((3,4))
print("b1=\n",b1)

#Create an array of ones
c1=np.ones((2,3,4),dtype=np.int16)
print("c1=\n",c1)
```

```
[[1 1 2 2]
 [3 6 4 5]]
a1=
 [0. 0. 0. 0. 0.]
b1=
 [[0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]]
c1=
 [[[1 1 1 1]
  [1 1 1 1]
  [1 1 1 1]]

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

```
#Create a new array of ones with the same shape and data type as a given input array.
b2 = np.ones_like(b1)
print(b2)
#Similarly, try with np.zeros_like(c1)
```

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


NumPy: Creating Array


```
#Create an array of evenly spaced values (step value)
d = np.arange(10,25,5)
print("d=\n",d)

#Create an array of evenly spaced values (number of samples)
e= np.linspace(0,1,11)
print("e=\n",e)

#Create a 2X2 identity matrix
f = np.eye(2)
print("f=\n",f)

#Create an array with random values
g=np.random.random((2,2))
print("g=\n",g)
```

```
d=
 [10 15 20]
e=
 [0.  0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1. ]
f=
 [[1. 0.]
 [0. 1.]]
g=
 [[0.86256337 0.37459907]
 [0.97249215 0.38316558]]
```

```
#Create array by drawing samples from a uniform distribution
print(np.random.uniform(0, 1, 10))
```

```
[0.47 0.72 0.79 0.65 0.4  0.18 0.32 0.25 0.46 0.3 ]
```

NumPy: Array Mathematics


```
#Setting precision
np.set_printoptions(precision=2)

a = np.array([1,2,3,4])
b = np.array([(1,1.5,2,2.5), (3,6,4,5)])

r = a - b #Subtraction
print(r)
print(np.subtract(a,b))
r= b + a  #Addition
print(r)
print(np.add(b,a))
r=a / b   #Division
print(r)
print(np.divide(a,b))

r=a * b   #Multiplication
print(r)
print(np.multiply(a,b))
```

```
[[ 0.   0.5  1.   1.5]
 [-2.  -4.  -1.  -1. ]]
[[ 0.   0.5  1.   1.5]
 [-2.  -4.  -1.  -1. ]]
[[2.  3.5 5.  6.5]
 [4.  8.  7.  9. ]]
[[2.  3.5 5.  6.5]
 [4.  8.  7.  9. ]]
[[1.   1.33 1.5  1.6 ]
 [0.33 0.33 0.75 0.8 ]]
[[1.   1.33 1.5  1.6 ]
 [0.33 0.33 0.75 0.8 ]]
[[ 1.  3.  6. 10.]
 [ 3. 12. 12. 20.]]
[[ 1.  3.  6. 10.]
 [ 3. 12. 12. 20.]]
```


NumPy: Array Mathematics


```
print(np.exp(b))#Exponentiation
print(np.sqrt(b))#Square root
print(np.sin(a))#Print sines of an array
print(np.cos(b))#Element-wise cosine
print(np.log(a))#Element-wise natural logarithm

e= np.full((2,2),5)
f= np.full((2,2),3)
print(e.dot(f))#Dot product

print("a=",a)
a_new=a
print("a_new=",a_new)
a_new[1]=9
print("a=",a)
a_new_copy=a.copy() #create a copy of array
a_new_copy[1]=2
print("a_new_copy=",a_new_copy)
print("a=",a)
```

```
[[  2.72   4.48   7.39  12.18]
 [ 20.09 403.43  54.6  148.41]]
[[1.   1.22 1.41 1.58]
 [1.73 2.45 2.   2.24]]
[ 0.84  0.91  0.14 -0.76]
[[ 0.54  0.07 -0.42 -0.8 ]
 [-0.99  0.96 -0.65  0.28]]
[0.   0.69 1.1  1.39]
```

```
[[30 30]
 [30 30]]
```

```
a= [1 2 3 4]
a_new= [1 2 3 4]
a= [1 9 3 4]
a_new_copy= [1 2 3 4]
a= [1 9 3 4]
```

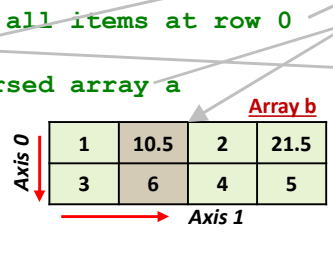
NumPy: Array Mathematics

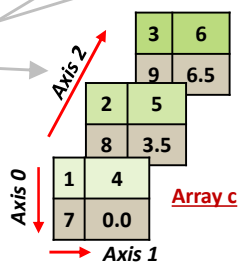

```
a = np.array([4,3,2,1])
b = np.array([(1,10.5,2,21.5), (3,6,4,5)])
c = np.array([[(1,2,3), (4,5,6)], [(7,8,9), (0.0,3.5,6.5)]],dtype='float32')
print(a)
print(b)

print(a.sum()) #Array-wise sum
print(a.min()) #Array-wise minimum value
print(b.max(axis=0)) #Maximum value axis 0
print(a.mean())#Mean
print(np.median(b)) #Median
print(np.corrcoef(a)) #Correlation coefficient
print(np.std(b)) #Standard deviation

a.sort()
print(a) #Sort an array
```

```
[4 3 2 1]
[[ 1.   10.5  2.   21.5]
 [ 3.    6.   4.    5. ]]
10
1
[ 3.   10.5  4.   21.5]
2.5
4.5
1.0
6.248749874974994
[1 2 3 4]
```


NumPy: Array Manipulation


```
#Subsetting
print(a)
print(a[2]) #Select the element at the 2nd index
print(b[1,2]) #Select the element at 2nd row 3rd column

#Slicing
print(a[0:2]) #Select items at index 0 and 1
print(b[0:2,1]) #Select items at rows 0 and 1 in column 1
print(b[:1]) #Select all items at row 0
print(c[1,:,:])
print(a[::-1]) #Reversed array a
```






```
#Transposing Array
print("c=\n",c)
print("c transpose (2,1,0)=\n",np.transpose(c,axes=(2,1,0)))
```

```
[1 2 3 4]
3
4.0
[1 2]
[10.5  6. ]
[[ 1.  10.5  2.  21.5]]
[[7.  8.  9. ]
 [0.  3.5 6.5]]
[4 3 2 1]
c=
 [[[1.  2.  3. ]
  [4.  5.  6. ]]

 [[7.  8.  9. ]
  [0.  3.5 6.5]]]
c transpose (2,1,0)=
 [[[1.  7. ]
  [4.  0. ]]

 [[2.  8. ]
  [5.  3.5]]

 [[3.  9. ]
  [6.  6.5]]]
```

NumPy: Array Manipulation


```
#Changing Array Shape
print("Flattened b=\n",b.ravel()) #Flatten the array; differ from b.flatten()
print("Reshaped b=\n",b.reshape(4,-1)) #Reshape, but don't change data

c.resize((2,6))
print("c=\n",c)
g=np.array([[.1,.2,.3,.4,.5,.6],
            [10,20,30,40,50,60]])
#Adding/Removing Elements
print(np.append(c,g))
print(np.append(c,g,axis=0))#Append along axis 0
print(np.append(c,g,axis=1))#Append along axis 1


print("'a'=\n",a)
#Insert item in an array
print("After inserting 5 =", np.insert(a, 1, 5))
#Delete an item
print("After removing 2nd element =", np.delete(a,[1]))
```

```
Flattened b=
 [ 1.  10.5  2.  21.5  3.   6.   4.   5. ]
Reshaped b=
 [[ 1.  10.5]
 [ 2.  21.5]
 [ 3.   6. ]
 [ 4.   5. ]]
c=
 [[1.  2.  3.  4.  5.  6. ]
 [7.  8.  9.  0.  3.5 6.5]]
[ 1.   2.   3.   4.   5.   6.   7.   8.   9.   0.   3.5  6.5  0.1  0.2
  0.3  0.4  0.5  0.6 10.  20.  30.  40.  50.  60. ]
[[ 1.   2.   3.   4.   5.   6. ]
 [ 7.   8.   9.   0.   3.5  6.5]
 [ 0.1  0.2  0.3  0.4  0.5  0.6]
 [10.  20.  30.  40.  50.  60. ]]
[[ 1.   2.   3.   4.   5.   6.   0.1  0.2  0.3  0.4  0.5  0.6]
 [ 7.   8.   9.   0.   3.5  6.5 10.  20.  30.  40.  50.  60. ]]
'a'=
 [1 2 3 4]
After inserting 5 = [1 5 2 3 4]
After removing 2nd element = [1 3 4]
```


NumPy: Array Manipulation


```
a = np.array([10,20,30])
b = np.array([(10,1.5,20.5), (3,6,5)])
c = np.array([(2.5,4.5,5.5), (3.5,6.5,5.5)])
d = np.arange(10,25,5)

print("a=\n",a)
print("d=\n",d)
#Concatenate arrays
print(np.concatenate((a,d),axis=0))
#Stack arrays vertically (row-wise)
print(np.vstack((a,d)))
#Create stacked column-wise arrays
print(np.column_stack((a,d)))
print(np.vstack((b,c))) #Stack arrays vertically (row-wise)
print(np.hstack((b,c))) #Stack arrays horizontally (column-wise)

#Splitting Arrays
print(np.hsplit(a,3)) #Split the array horizontally
print(np.vsplit(c,2)) #Split the array vertically
```

```
a=
 [10 20 30]
d=
 [10 15 20]
[10 20 30 10 15 20]
[[10 20 30]
 [10 15 20]]
[[10 10]
 [20 15]
 [30 20]]
[[10.   1.5 20.5]
 [ 3.   6.   5. ]
 [ 2.5  4.5  5.5]
 [ 3.5  6.5  5.5]]
[[10.   1.5 20.5  2.5  4.5  5.5]
 [ 3.   6.   5.   3.5  6.5  5.5]]
[array([10]), array([20]), array([30])]
[array([[2.5, 4.5, 5.5]]), array([[3.5, 6.5, 5.5]])]
```

User-defined functions on NumPy arrays


```
#User-defined function on NumPy array
def factorial(n):
    if n == 0 or n == 1:
        return 1
    else:
        f = 1
        for i in range(2, n+1):
            f = f * i
        return f

import numpy as np
print("Factorial of 5 is",factorial(5))
a = np.array(range(10))
print("a=", a)
b = factorial(a)
print("b=", b)
```

```
Factorial of 5 is 120
a= [0 1 2 3 4 5 6 7 8 9]
Traceback (most recent call last):

  Cell In[125], line 14
    b = factorial(a)

  Cell In[125], line 2 in factorial
    if n == 0 or n == 1:

ValueError: The truth value of an
array with more than one element is
ambiguous. Use a.any() or a.all()
```


User-defined functions on NumPy arrays


```
#User-defined function on NumPy array
def factorial(n):
    if n == 0 or n == 1:
        return 1
    else:
        f = 1
        for i in range(2, n+1):
            f = f * i
        return f

import numpy as np
print("Factorial of 5 is",factorial(5))
a = np.array(range(10))
print("a=", a)
compFactorial = np.frompyfunc(factorial, 1, 1)
b = compFactorial(a)
print("b=", b)
```

```
Factorial of 5 is 120
a= [0 1 2 3 4 5 6 7 8 9]
b= [1 1 2 6 24 120 720 5040 40320 362880]
```

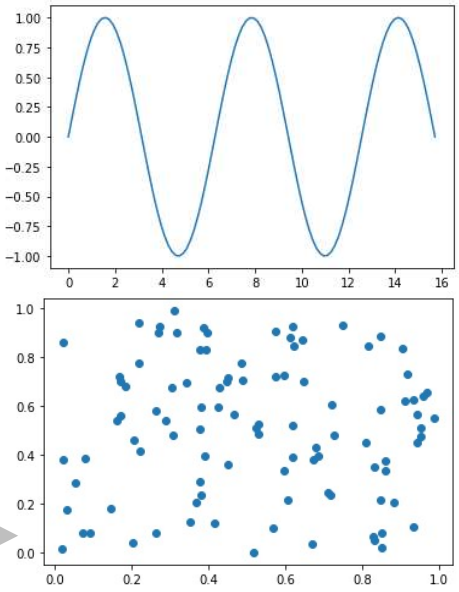
15

matplotlib

Matplotlib: Creating Plots

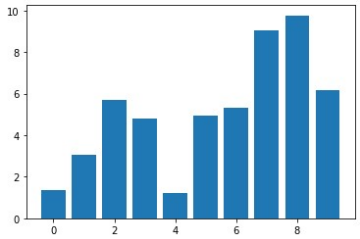
16

```
#Required import statements
import numpy as np
import matplotlib.pyplot as plt

#Preparing Data
X = np.linspace(0, 5*np.pi, 100)
Y = np.sin(X)

#Plotting
fig, ax = plt.subplots()
ax.plot(X, Y)
plt.show()

fig, ax = plt.subplots()
X = np.random.uniform(0, 1, 100)
Y = np.random.uniform(0, 1, 100)
ax.scatter(X, Y)
plt.show()
```

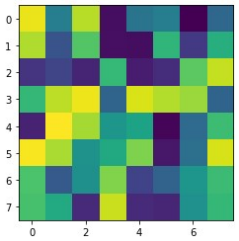


Matplotlib: Creating Plots

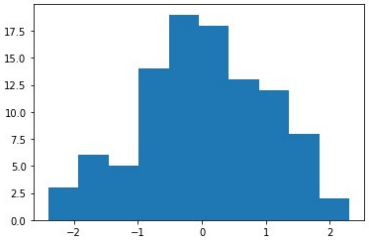

```
fig, ax = plt.subplots()
X = np.arange(10)
Y = np.random.uniform(1, 10, 10)
ax.bar(X, Y)
plt.show()
```



```
fig, ax = plt.subplots()
Z = np.random.uniform(0, 1, (8, 8))
ax.imshow(Z)
plt.show()
```



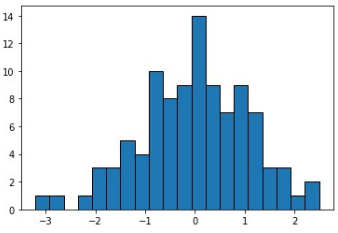
```
fig, ax = plt.subplots()
Z = np.random.normal(0, 1, 100)
ax.hist(Z)
plt.show()
```



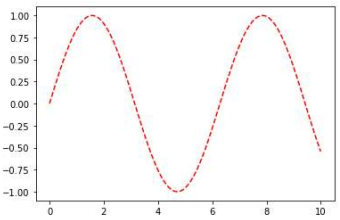
```
fig, ax = plt.subplots()
Z = np.random.uniform(0, 1, 4)
ax.pie(Z)
plt.show()
```

Matplotlib: Creating Plots

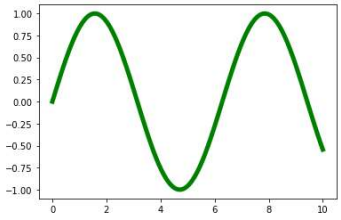

```
fig, ax = plt.subplots()
Z = np.random.normal(0, 1, 100)
ax.hist(Z,edgecolor='k',bins=20)
plt.show()
```



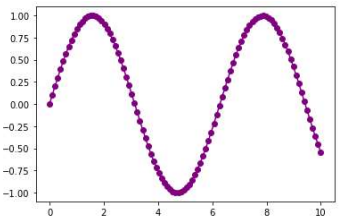
```
fig, ax = plt.subplots()
X = np.linspace(0, 10, 100)
Y = np.sin(X)
ax.plot(X, Y, linestyle='--',color='r')
```



```
fig, ax = plt.subplots()
X = np.linspace(0, 10, 100)
Y = np.sin(X)
ax.plot(X, Y, linewidth=5,color='g')
```

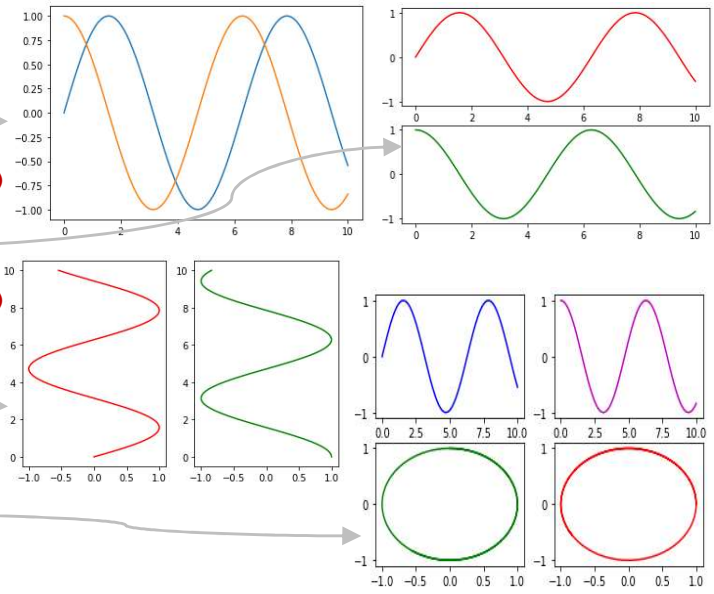


```
fig, ax = plt.subplots()
X = np.linspace(0, 10, 100)
Y = np.sin(X)
ax.plot(X, Y, marker='o', color='purple')
```



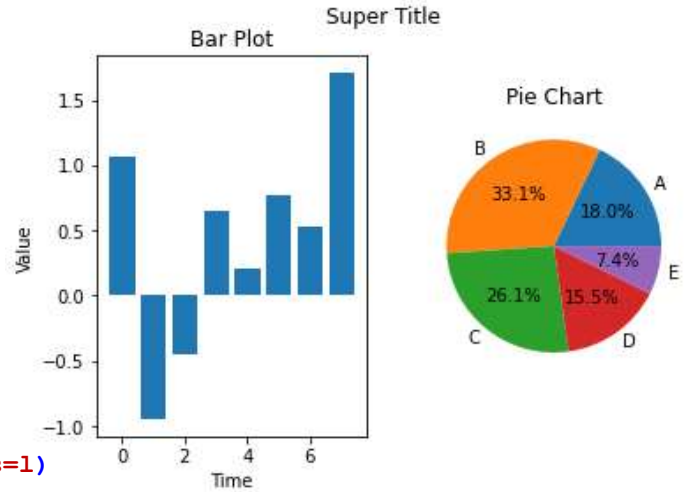
https://matplotlib.org/stable/gallery/lines_bars_and_markers/linestyles.html
https://matplotlib.org/3.1.0/api/markers_api.html

Matplotlib: Organizing Plots



```
fig, ax = plt.subplots()
X = np.linspace(0, 10, 100)
Y1, Y2 = np.sin(X), np.cos(X)
ax.plot(X, Y1, X, Y2)

fig, (ax1, ax2) = plt.subplots(2, 1)
ax1.plot(X, Y1, color='r')
ax2.plot(X, Y2, color='g')

fig, (ax1, ax2) = plt.subplots(1, 2)
ax1.plot(Y1, X, color='r')
ax2.plot(Y2, X, color='g')

fig, ax = plt.subplots(2, 2)
ax=ax.flatten()
ax[0].plot(X, Y1, color='b')
ax[1].plot(X, Y2, color='m')
ax[2].plot(Y1, Y2, color='g')
ax[3].plot(Y2, Y1, color='r')
```

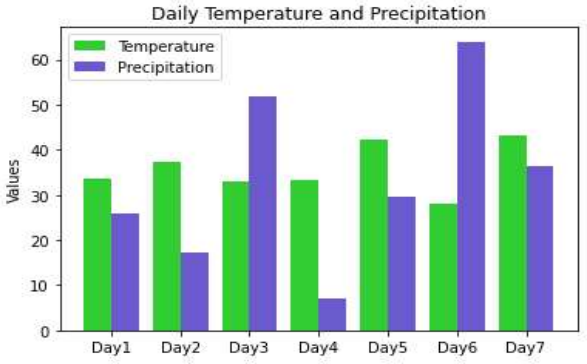

Matplotlib: Plot Labelling




```
X=np.array(range(8))
Y=np.random.normal(0, 1, 8)
Z=np.random.uniform(0, 1, 5)

fig, ax = plt.subplots(1,2)
fig.suptitle('Super Title')

ax[0].set_title('Bar Plot')
ax[0].bar(X, Y)
ax[0].set_ylabel('Value')
ax[0].set_xlabel('Time')
ax[1].set_title('Pie Chart')
l=['A','B','C','D','E']
ax[1].pie(Z,autopct='%1.1f%%',labels=l)

plt.show()
```

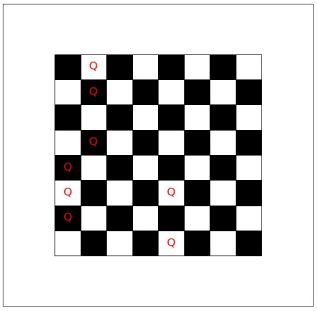
Matplotlib: Plot Labelling and Saving


```
X= np.array(range(7))
l=['Day1', 'Day2', 'Day3', 'Day4', 'Day5', 'Day6', 'Day7']
Y= np.random.uniform(20,45,7)
Z= np.random.normal(50,20,7)

w=0.4
fig, ax = plt.subplots()
ax.bar(X - w/2, Y, width=w, label='Temperature',color='limegreen')
ax.bar(X + w/2, Z, width=w, label='Precipitation',color='slateblue')

ax.set_xticks(X)
ax.set_xticklabels(l)
ax.set_ylabel('Values')
ax.set_title('Daily Temperature and Precipitation')
ax.legend()

plt.show()


#Saving Figure
fig.savefig('figure_name.png', dpi=300)
fig.savefig('figure_name.pdf')
```



Matplotlib: Patches


```
import matplotlib.pyplot as plt
import matplotlib.patches as patches
import numpy as np
fig, ax= plt.subplots(figsize=(10,10))
for i in range(8):
   for j in range(8):
      if (i+j)%2==0:
         bfill = patches.Rectangle((i, j), 1 , 1, linewidth=1, edgecolor='black', facecolor='white')
      else:
         bfill = patches.Rectangle((i, j), 1 , 1, linewidth=1, edgecolor='black', facecolor='black')
      ax.add_patch(bfill)
ax.set_xlim(-2,10)
ax.set_ylim(-2,10)
ax.set_xticks([])
ax.set_yticks([])
for i in range(8):
    r1=np.random.random()
    for j in range(8):
        r2=np.random.random()
        if r1>0.7 and r2>0.7:
            ax.text(i+0.5,j+0.4, "Q", ha='center',fontsize=20,color="red")
plt.show()
```

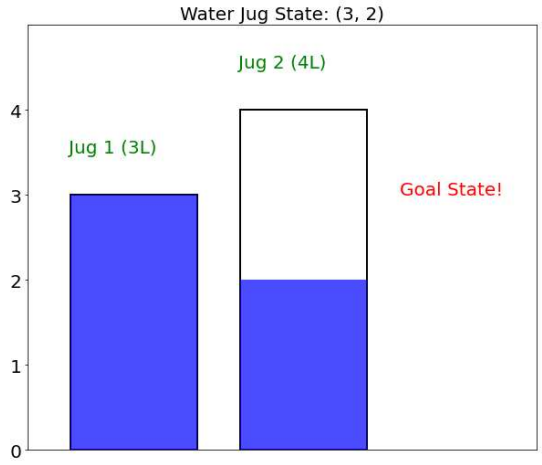


Home Assignment

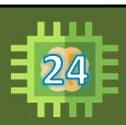
Use *basic concept of Python Programing* and the concepts from `matplotlib.patches` to define a function, named as "jugplot", that takes four input parameters/arguments, namely

- **Capacity of Jug1,**
- **Capacity of Jug2,**
- **Current Water Content of Jug1, and**
- **Current Water Content of Jug2,**

to produce a figure showing the status of both the water jugs. Moreover, the text "Goal State" is added on the plot in case it shows 2L water in Jug2



Example output for the following function call: `jugplot(3,4,3,2)`



Questions?