Name: Kseniia Shilova

Email: kshilova3@gatech.edu

Current position: PhD in Quantitative Biosciences. I will do two rotations this term. One of them is in Professor Hannah Choi's lab (mathematical/computational neuroscience), and the second one is in Professor Patrick McGrath's lab (I will work on computer vision tasks mostly).

Previous experience with Python: I have a bachelor degree in Software Engineering, and I worked in the algebraic topology lab, where I did Topological Data Analysis of brain (drosophila's connectome and AD-related fMRI) and genes expression data. I am good at Python, C++, C#, Java, and I have some experience in R. I had many courses related to Data Analysis, so I worked with sklearn, PyTorch, tensorflow, and visualization packages (matplotlib, seaborn). I also worked with graph analysis packages (NetworkX, igraph) and GUDHI package for TDA (topological data analysis). I had a minor in Bioinformatics, so I am good at Linux, I worked with text files a lot (with Python also).

Previous experience with data analysis: I did Topological Data Analysis, I did dimensionality reduction of data with topology preserving methods, I have experience in Natural Language Processing and Computer Vision a little bit.

Project ideas:
In general, I would like to work with brain data. For example, my first idea is to compare dimensionality reduction methods for functional brain data. The result is the comparison table for different networks and the methods, which are better for them.
My second idea is to analyze behavioral data. For example, we can cluster types of actions, which are based on 3D movements of animals. It could be based on the existing data. As an example, it is possible to compare behavioral maps (clusters) for different time points of an animal's development.

Student: Kseniia Shilova

0. Introduction to Python and NumPy

Instructor: Eva Dyer, BMED 6517

---

1. Python Basics

A **cell** is a container for code to be executed by the python **kernel**. When you run the cell, its output will be displayed below. You can click the run button or press `Shift+Enter`.

1.1 Basic variable manipulation

In [1]:
```
print("Hello world!")
```

```
Hello world!
```

The equal sign `=` is used to assign a value to a variable.

In [2]:
```
a = 5 ** 2  # 5 squared
print(a)

a = (a - 10) * 2
print(a)
```

```
25
30
```

1.2. `for` Statements

The `for` statement is used to iterate over the indented code.

Use `range()` to iterate over a sequence of numbers for example. Note that `range(n)` goes from `0` to `n-1`.

In [3]:
```
cumsum = 0  # holds cumulative sum

for i in range(3):
  print(i)
  cumsum = cumsum + i
  print('Cumulative Sum:', cumsum)
```

```
0
Cumulative Sum: 0
1
Cumulative Sum: 1
2
Cumulative Sum: 3
```

1.3. Lists

Lists are used to group together multiple values. They might contain items of different types, but usually the items all have the same type.

`len()` is used to access the length of a list.

In [4]:
```
squares = [1, 4, 9, 16, 25]
print('This list has', len(squares), 'elements.')
```

```
This list has 5 elements.
```

To access an element in the list, use indexing. Note that the first element in a list has index `0`.

In [5]:
```
print('first element:', squares[0])  # indexing returns the item
print('third element:', squares[2])
print('last element:', squares[4])
print('also last element:', squares[-1])
```

```
first element: 1
third element: 9
last element: 25
also last element: 25
```

Lists can be sliced. `squares[a:b]` will return a new list with the elements between indices `a` and `b-1`.

In [6]:
```
print(squares[1:3])
```

```
[4, 9]
```

1.4. Defining functions

When certain blocks of code are to be used multiple times, defining functions can be useful. In general, it is helpful to break down the code into small an modular components.

Below is an example of a function that takes in arguments and returns a value.

In [7]:
```
def compute_sum(l):
    sum = 0
    for i in range(len(l)):
        sum += l[i]
    return sum

print('The sum of', squares, 'is', compute_sum(squares))
```

```
The sum of [1, 4, 9, 16, 25] is 55
```

1.5. Python Packages

Python comes with a library of standard modules, like `math` for example. The module itself, or a specific function can be loaded in.

In [8]:
```
import math

print(math.cos(0))
```

```
1.0
```

In [9]:
```
from math import cos, pi

print(cos(pi))
```

```
-1.0
```

More modules can be installed through `pip`, the package installer for Python. We can install the `art` for example and then load it in to make cool ASCII art.

In [10]:
```
!pip install art #<- note that ! lets us run a bash command
```

```
Collecting art
  Downloading art-6.1-py3-none-any.whl (599 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 599.8/599.8 kB 2.6 MB/s eta 0:00:00
Installing collected packages: art
Successfully installed art-6.1
```

In [11]:
```
from art import tprint, art

# print pretty text
tprint("\(^-^)/")
```

```
__      __ /\         /\ __      __
\ \    / /|/\|       |/\|\ \    / /
 \ \  | |      _____      | |  / /
  \ \ | |     |_____|     | | / /
   \_\| |                 | |/_/
       \_\               /_/
```

In [12]:
```
# draw 3 random Strings. (Hint: try running the cell multiple times.)
for i in range(3):
    print(art("random"))
```

```
>>-;;;------;;-->
<><
(===||:::::::::::::::>
```

1.6. Example: Creating a matrix

A list can contain any type of object, that includes lists themselves. These are called nested lists. Let's create a 3x4 matrix.

In [13]:
```
matrix = [[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0]]

print('first row:', matrix[0])
print('element at row 0 and column 2:', matrix[0][2])
```

```
first row: [0, 0, 0, 0]
element at row 0 and column 2: 0
```

Let's add 1s to the main diagonal of the matrix.

In [14]:
```
for i in range(3):
    for j in range(4):
        if i == j:
            matrix[i][j] = 1

print(matrix)
```

```
[[1, 0, 0, 0], [0, 1, 0, 0], [0, 0, 1, 0]]
```

1.7. More resources:

- https://docs.python.org/3/tutorial/introduction.html
- https://docs.python.org/3/tutorial/

2. Numpy Basics and Arrays

2.1. Matrices (Numpy arrays)

- A Numpy array is the Python data type for storing/manipulating multi-dimensional matrices.
- Matrices are an extreme example of structured data. Data is accessed by providing indices for each dimension. Indices must be integers, and all data must be numerical.
- Matrices do not have a built-in schema system, so it must be managed separately. We will briefly explore how to work with matrices in Python and how to use them for data storage.

In [15]:
```
import numpy as np  # <- very common shorthand for numpy
```

2.2. Creating `numpy` arrays

There are a number of ways to initialize new numpy arrays, for example from

- a Python list or tuples
- using functions that are dedicated to generating numpy arrays, such as `arange`, `linspace`, etc.

- reading data from files

In [16]:
```
# a vector: the argument to the array function is a Python list
v = np.array([1,2,3,4])

print(v)
```

```
[1 2 3 4]
```

In [17]:
```
# a matrix: the argument to the array function is a nested Python list
M = np.array([[1, 2], [3, 4]])

print(M)
```

```
[[1 2]
 [3 4]]
```

The difference between the `v` and `M` arrays is only their shapes. We can get information about the shape and size of an array by using the `shape` and `size` properties.

In [18]:
```
print('v: number of dimensions=', v.ndim, ', shape=', v.shape)
print('M: number of dimensions=', M.ndim, ', shape=', M.shape)
```

```
v: number of dimensions= 1 , shape= (4,)
M: number of dimensions= 2 , shape= (2, 2)
```

Arrays are similar to lists, but they must contain a single type:

In [19]:
```
M[0,0] = 10
print(M)
```

```
[[10  2]
 [ 3  4]]
```

If we want, we can explicitly define the type of the array data when we create it, using the `dtype` keyword argument:

In [20]:
```
v = np.array([1, 2, 3, 4], dtype=np.uint8)
print(v, v.dtype)

v = np.array([1, 2, 3, 4], dtype=np.float)
print(v, v.dtype)
```

```
[1 2 3 4] uint8
[1. 2. 3. 4.] float64
```

```
<ipython-input-20-90ebef2c1699>:4: DeprecationWarning: `np.float` is a deprecated alias for the builtin `float`. To sil
ence this warning, use `float` by itself. Doing this will not modify any behavior and is safe. If you specifically want
ed the numpy scalar type, use `np.float64` here.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/release/1.20.0-notes.html#deprecatio
ns
  v = np.array([1, 2, 3, 4], dtype=np.float)
```

2.3. Creating arrays with functions

It is often more efficient to generate large arrays instead of creating them from lists. There are a few useful functions for this in numpy.

`np.arange` creates a range with a specified step size (endpoints not included)

In [21]:
```
x = np.arange(0, 4, 0.5) # arguments: start, stop, step
print(x)
```

```
[0.  0.5 1.  1.5 2.  2.5 3.  3.5]
```

`np.linspace` creates a range with a specified number of points (endpoints are included)

In [22]:
```
x = np.linspace(0,10,5)
print(x)
```

```
[ 0.   2.5  5.   7.5 10. ]
```

`np.zeros` creates a matrix of zeros.

`np.ones` creates a matrix of ones.

`np.eye` creates an identity matrix.

In [23]:
```
print('\n 2d-Matrix of shape (2,3) filled with zeros\n', np.zeros((2,3)))
print('\n 3d-Matrix of shape (2,2,2) filled with ones\n', np.ones((2, 3, 4)))
print('\n Identity matrix of shape (3,3)\n', np.eye(3))
```

```
2d-Matrix of shape (2,3) filled with zeros
[[0. 0. 0.]
[0. 0. 0.]]

3d-Matrix of shape (2,2,2) filled with ones
[[[1. 1. 1. 1.]
  [1. 1. 1. 1.]
  [1. 1. 1. 1.]]

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

Identity matrix of shape (3,3)
[[1. 0. 0.]
[0. 1. 0.]
[0. 0. 1.]]
```

2.4. Manipulating arrays

Once we generate `numpy` arrays, we need to interact with them. This involves a few operations:

- indexing - accessing certain elements
- index "slicing" - accessing certain subsets of elements
- fancy indexing - combinations of indexing and slicing

This is not very different from Matlab.

We can index elements in an array using square brackets and indices:

In [24]:
```
# v is a vector, and has only one dimension, taking one index
print(v[0])
# M is a matrix, or a 2 dimensional array, taking two indices
print(M[1,1])
# If an index is ommitted then the whole row is returned
print(M[1])
```

```
1.0
4
[3 4]
```

We can assign new values to elements or rows in an array using **indexing**:

In [25]:
```
M[:,1] = -1
print(M)
```

```
[[10 -1]
 [ 3 -1]]
```

Index slicing is the name for the syntax `M[lower:upper]` to extract a subset of an array:

In [26]:
```
A = np.arange(1,20)
print(A)
print(A[1:8])
```

```
[ 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19]
[2 3 4 5 6 7 8]
```

Fancy indexing is the name for when an array or list is used in-place of an index:

In [27]:
```
R = np.eye(4)
print(R, '\n')

row_indices = np.array([1, 3])
print(R[row_indices])
```

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

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

2.5. Transposing arrays

Arrays can easily be transposed with `.T`.

In [28]:
```
M = np.array([[1,2,3], [2,1,4]])
print(M)
print('shape', M.shape)

print('\n')

print(M.T)
print('shape', M.T.shape)
```

```
[[1 2 3]
 [2 1 4]]
shape (2, 3)


[[1 2]
 [2 1]
 [3 4]]
shape (3, 2)
```

2.6. Computing statistics

Mean: $\mu = E[X]$

Variance: $\sigma^2 = E[(X - \mu)^2] = E[X^2] - (E[X])^2$

In [29]:
```
v = np.array([1, 2, 0, 4, 10, 8])

print('max:', np.max(v))
print('sum:', np.sum(v))
print('mean:', np.mean(v))
print('standard deviation:', np.std(v))
```

```
max: 10
sum: 25
mean: 4.166666666666667
standard deviation: 3.6704525909242065
```

2.7. Example: Computing the L2 distance between two vectors

In [30]:
```
v = np.array([1, 2, 0, 4, 10, 8])
w = np.array([2, 1, 2, 7, 8, 9])

dist = np.sqrt(np.sum((v - w)**2))
print(dist)
```

```
4.47213595499958
```

Challenge: What is the L1-distance between v and w?

In [31]:
```
# Add code to compute the L1 distance - Are there other functions that can compute general Lp-norms?

# 1. L1 distance
```

```
dist_L1 = np.sum(np.abs((v - w)))
print(f'Simple formula with the help of numpy: Manhattan Distance (L1) = {dist_L1}', end='\n\n')

# 2. General function for Lp-norm
def my_norm_function(x, p):
    return (np.sum([np.abs(i**p) for i in x]))**(1/p)
print(f'My function: Manhattan Distance (L1) = {my_norm_function((v-w), 1)}')
for j in range(2,6):
    print(f'My function: L{j}-norm = {my_norm_function((v-w), j)}')
print('\n')

# 3. Numpy linalg norm function
for j in range(1,6):
    print(f'Numpy linalg norm function: L{j}-norm = {np.linalg.norm((v-w), ord=j)}')
print('\n')

# 4. Scipy linalg norm function
import scipy
for j in range(1,6):
    print(f'Scipy linalg norm function: L{j}-norm = {scipy.linalg.norm((v-w), ord=j)}')
print('\n')
```

```
Simple formula with the help of numpy: Manhattan Distance (L1) = 10

My function: Manhattan Distance (L1) = 10.0
My function: L2-norm = 4.47213595499958
My function: L3-norm = 3.583047871015946
My function: L4-norm = 3.281818034911291
My function: L5-norm = 3.1497228331682314


Numpy linalg norm function: L1-norm = 10.0
Numpy linalg norm function: L2-norm = 4.47213595499958
Numpy linalg norm function: L3-norm = 3.583047871015946
Numpy linalg norm function: L4-norm = 3.281818034911291
Numpy linalg norm function: L5-norm = 3.1497228331682314


Scipy linalg norm function: L1-norm = 10.0
Scipy linalg norm function: L2-norm = 4.47213595499958
Scipy linalg norm function: L3-norm = 3.583047871015946
Scipy linalg norm function: L4-norm = 3.281818034911291
Scipy linalg norm function: L5-norm = 3.1497228331682314
```

2.8. Example: Computing powers of 2

In [32]:
```
pow_two = 2 ** np.arange(4, 12)
print(pow_two)
```

```
[  16   32   64  128  256  512 1024 2048]
```

More resources

- https://numpy.org/doc/stable/user/absolute_beginners.html

3. Matrix-Vector Operations

You should already be famililar with linear algebra, but we will briefly review the basics and show how it works in `numpy` by covering the following:

Formulating your code as matrix-matrix and matrix-vector operations in Numpy will make it much more efficient. We will briefly cover syntax for:

- scalar*vector
- scalar*matrix
- matrix*vector
- matrix*matrix
- inverse
- eigendecomposition

`numpy` notes:

- reshaping and resizing arrays
- boolean and comparison operators on arrays

3.1. Scalar-array operations

We can use the usual arithmetic operators to multiply, add, subtract, and divide arrays with scalar numbers.

In [33]:
```
v = np.arange(0, 5)
print('v:', v)

print('v*2:', v*2)

print('v+2:', v+2)
```

```
v: [0 1 2 3 4]
v*2: [0 2 4 6 8]
v+2: [2 3 4 5 6]
```

In [34]:
```
M = np.ones((2,2))
print('M:\n', M)
print('M*2:\n', M*2)
print('M+2:\n', M+2)
```

```
M:
 [[1. 1.]
 [1. 1.]]
M*2:
 [[2. 2.]
 [2. 2.]]
M+2:
 [[3. 3.]
 [3. 3.]]
```

3.2. Element-wise array-array operations

When we add, subtract, multiply and divide arrays with each other, the default behaviour is **element-wise** operations. This is different from Matlab!

In [35]:
```
v = np.arange(2,6)
print('v:', v)
print('v.v:', v*v)
print('v/v:', v/v)

M = np.array([[1,2],[3,4]])
print('M:\n', M)
print('M.M:\n', M*M)
```

```
v: [2 3 4 5]
v.v: [ 4  9 16 25]
v/v: [1. 1. 1. 1.]
M:
 [[1 2]
 [3 4]]
M.M:
 [[ 1  4]
 [ 9 16]]
```

3.3. Matrix algebra

What about matrix mutiplication?

- use the `dot` function

In [36]:
```
A = np.eye(3,3)
v = np.array([1,2,3])

print('A:\n', A)
print('v:', v)

print('A*v.T:', np.dot(A, v.T))
print('A*A:\n', np.dot(A,A))
print('v*v:', np.dot(v.T,v))
```

```
A:
 [[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
v: [1 2 3]
A*v.T: [1. 2. 3.]
A*A:
 [[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
v*v: 14
```

3.4. Common matrix operations

We can easily calculate the inverse using `inv`

In [37]:
```
A = np.array([[-1, 2], [3, -1]])
print('A:\n', A)
print('inv(A):\n', np.linalg.inv(A))
```

```
A:
 [[-1  2]
 [ 3 -1]]
inv(A):
 [[0.2 0.4]
 [0.6 0.2]]
```

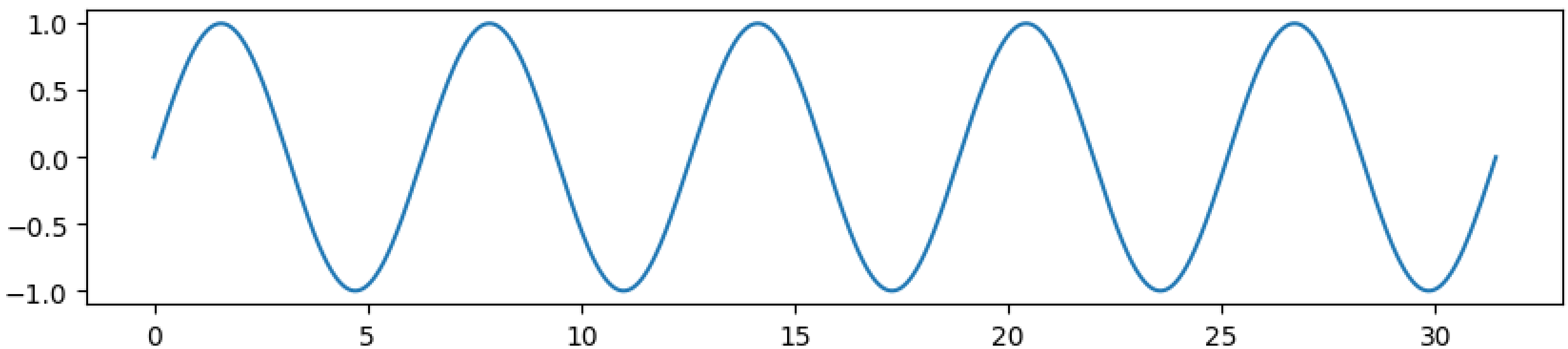
4. Visualization with matplotlib

Matplotlib is a library for making 2D plots in Python. It offers several kinds of plots. Here we highlight some of the most commonly used plots.

In [38]:
```
import matplotlib.pyplot as plt
```

4.1. Line plot

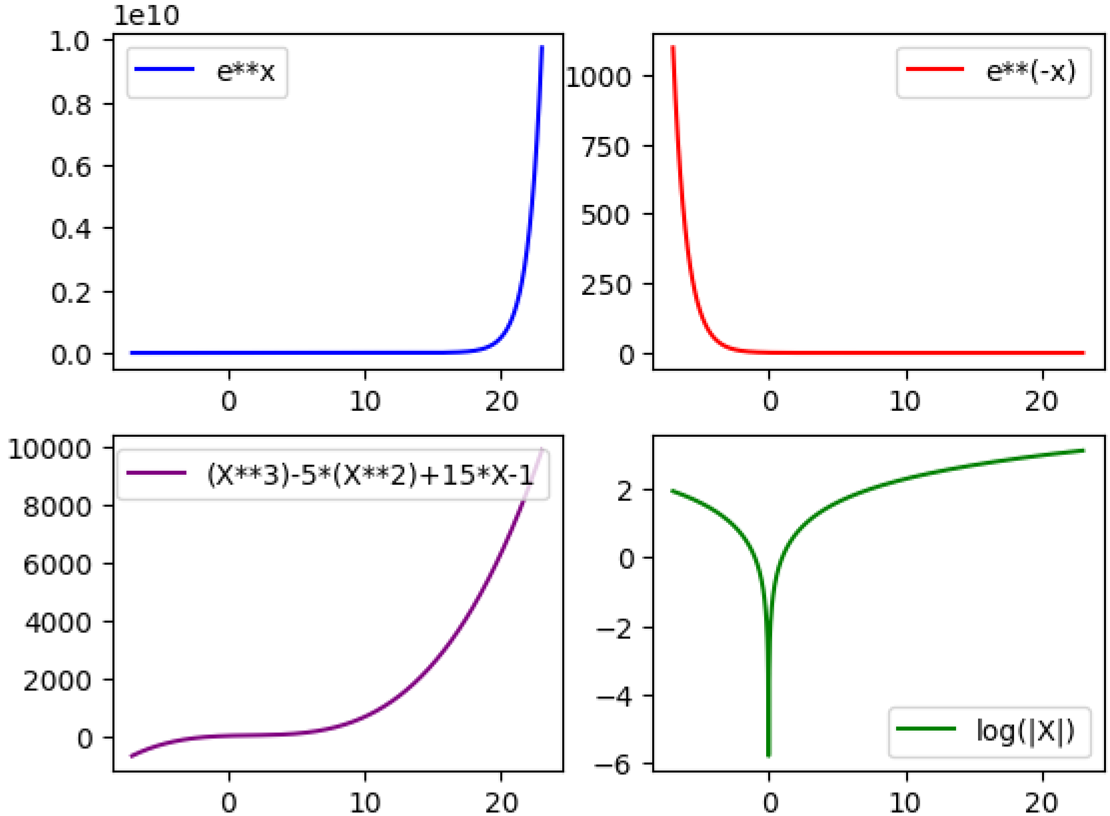
In [39]:
```
X = np.linspace(0, 10*np.pi, 1000)
Y = np.sin(X)

plt.figure(figsize=(10, 2))
plt.plot(X,Y)
plt.show()
```



Spend a few moments thinking about different functions that you could generate (linear? exponential?) and create a few subplots in the cell below.

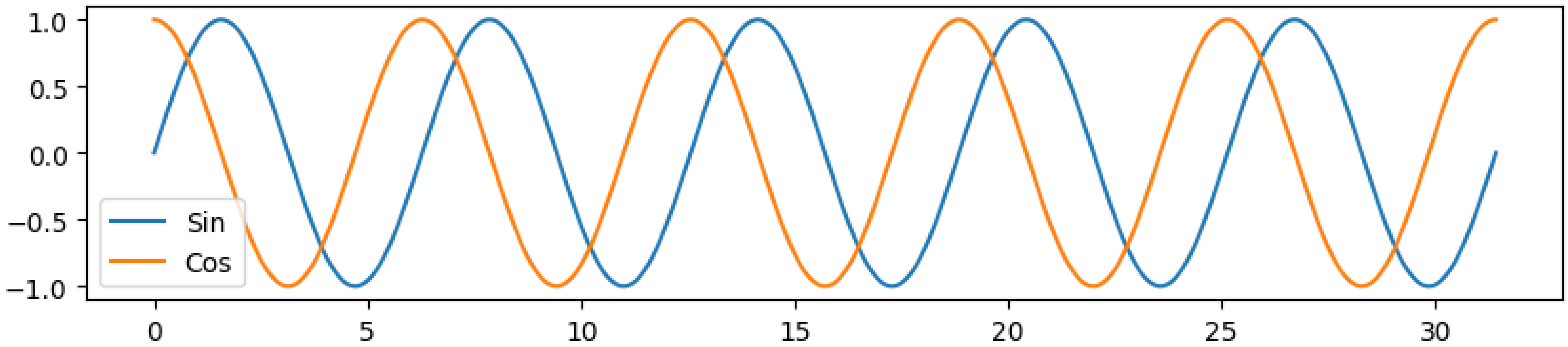
Check out these examples: https://matplotlib.org/stable/gallery/subplots_axes_and_figures/subplots_demo.html

In [40]:
```
# Plot different functions using subplots

X = np.linspace(-7, 23, 1000) # just random interval

fig, axs = plt.subplots(2, 2)
axs[0, 0].plot(X, np.e**(X), color='blue', label='e**x') #exp
axs[0, 1].plot(X, np.e**(-X), color='red', label='e**(-x)') #exp
axs[1, 0].plot(X, (X**3)-5*(X**2)+15*X-1, color='purple',
               label='(X**3)-5*(X**2)+15*X-1') #polynomial
axs[1, 1].plot(X, np.log(np.abs(X)), color='green', label='log(|X|)') #log

for ax in axs.flat:
    ax.legend()
```



You can plot several data on the same figure. Make sure to add a legend!

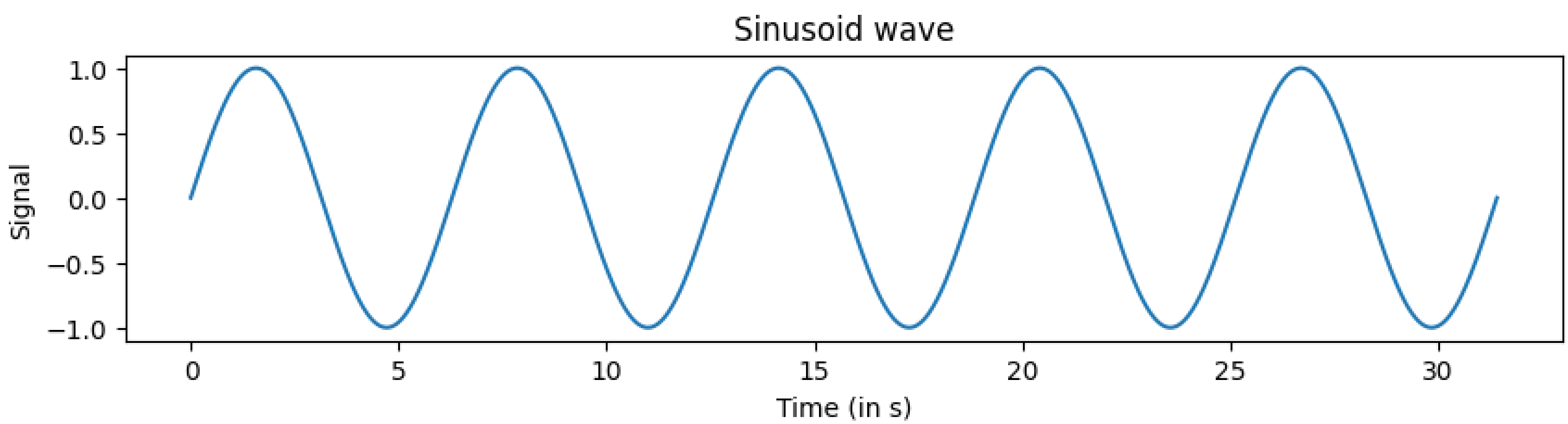
In [41]:
```
X = np.linspace(0, 10*np.pi, 1000)
Y1 = np.sin(X)
Y2 = np.cos(X)

plt.figure(figsize=(10, 2))
plt.plot(X,Y1)
plt.plot(X,Y2)
plt.legend(['Sin', 'Cos'])
plt.show()
```



4.2. Labeling a figure

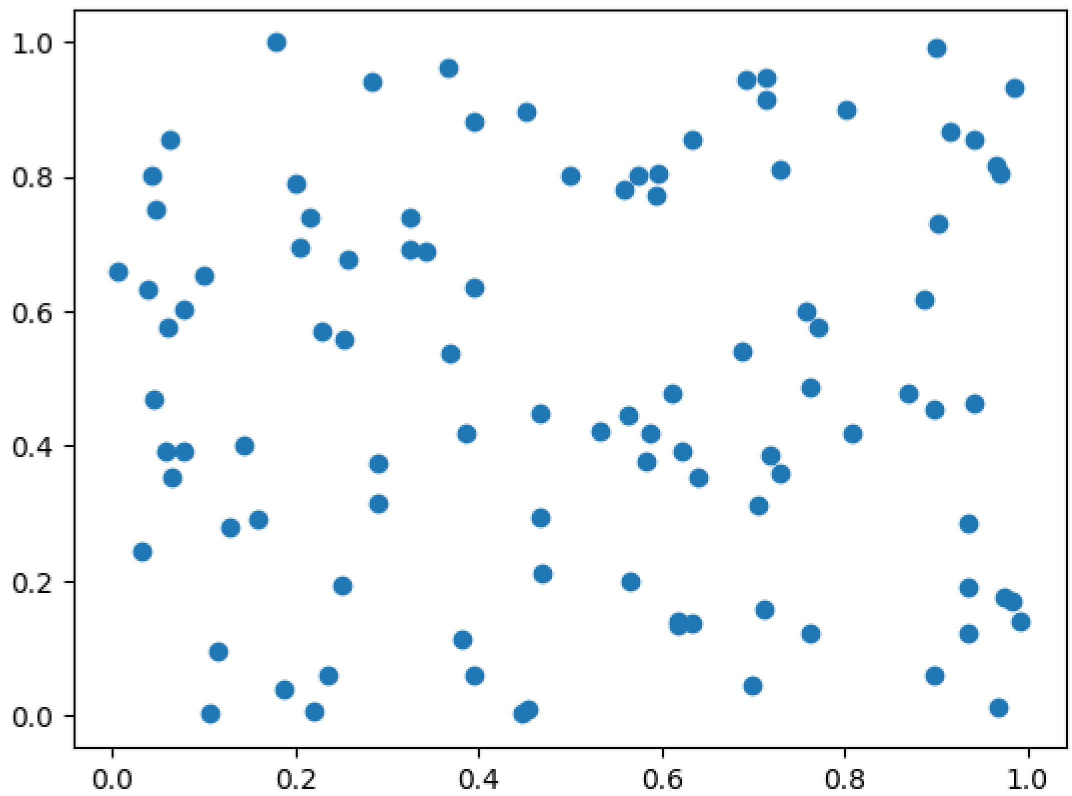
Making a nice figure is awesome, but without labels, your readers cannot appreciate it.

In [42]:
```
plt.figure(figsize=(10, 2))
plt.plot(X,Y)
plt.title("Sinusoid wave")
plt.ylabel("Signal")
plt.xlabel("Time (in s)")
plt.show()
```



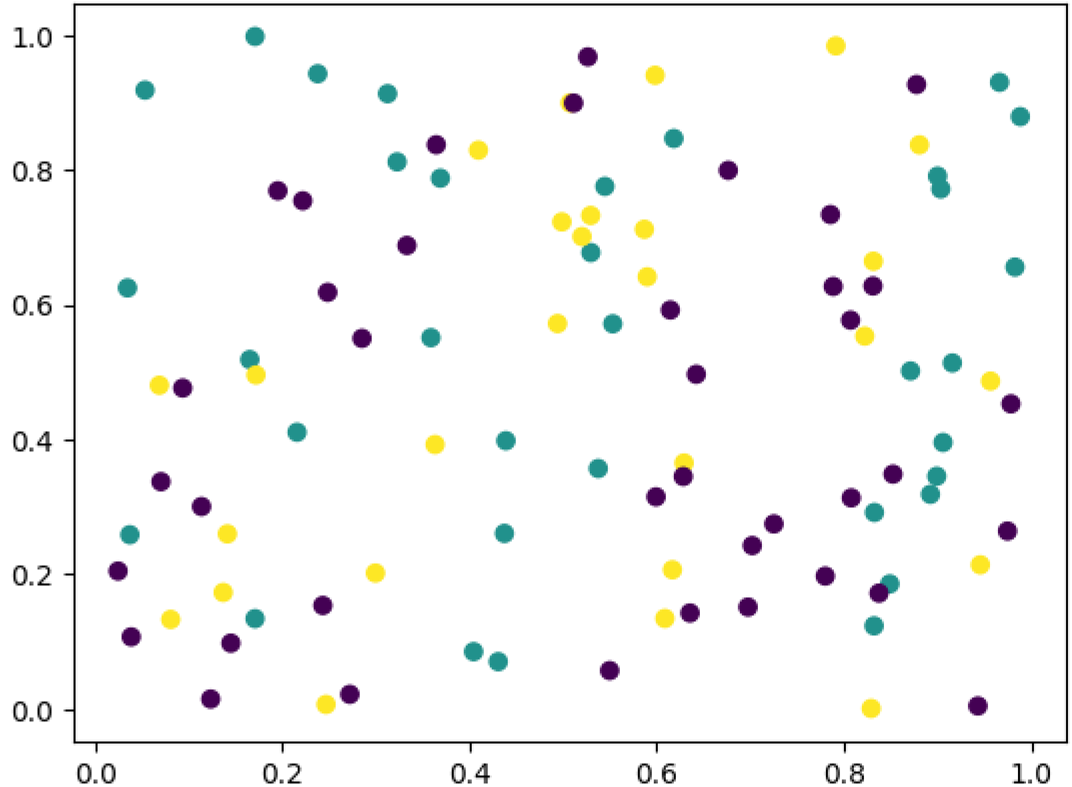
4.3. Scatter plot

In [43]:
```
X = np.random.uniform(0, 1, 100)
Y = np.random.uniform(0, 1, 100)
plt.scatter(X,Y)
plt.show()
```



We can add color to a scatter plot

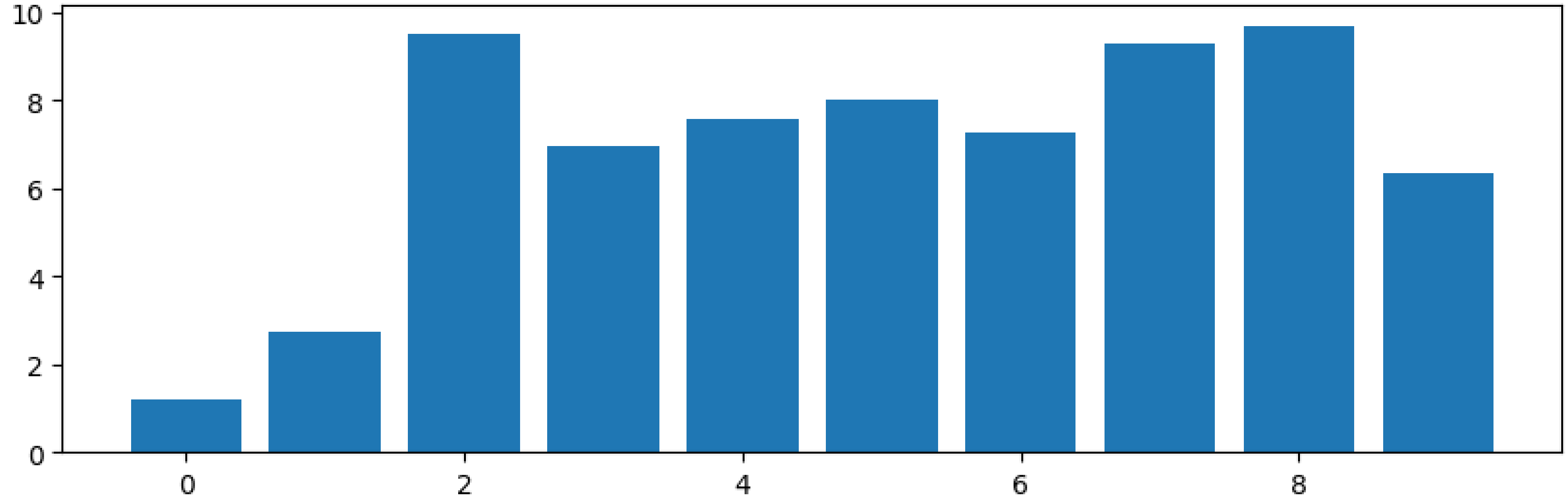
In [44]:
```
X = np.random.uniform(0, 1, 100)
Y = np.random.uniform(0, 1, 100)
color = np.random.randint(0, 3, 100)
plt.scatter(X,Y, c=color)
plt.show()
```



4.4. Bar plot

In [45]:

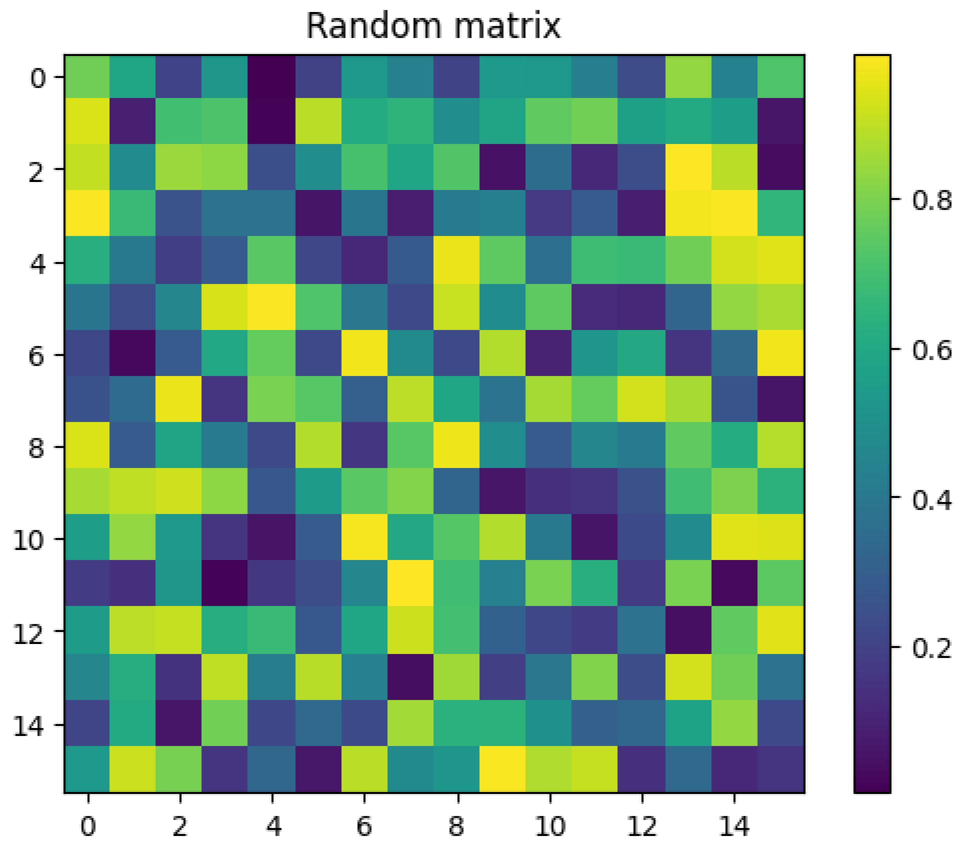
```
X = np.arange(10)
Y = np.random.uniform(1, 10, 10)
plt.figure(figsize=(10, 3))
plt.bar(X, Y)
plt.show()
```



4.5. Matrix visualization

4.5.1 Uniform distribution

In [46]:
```
Z = np.random.uniform(0, 1, (16, 16))
plt.imshow(Z)
plt.title('Random matrix')
plt.colorbar()
plt.show()
```

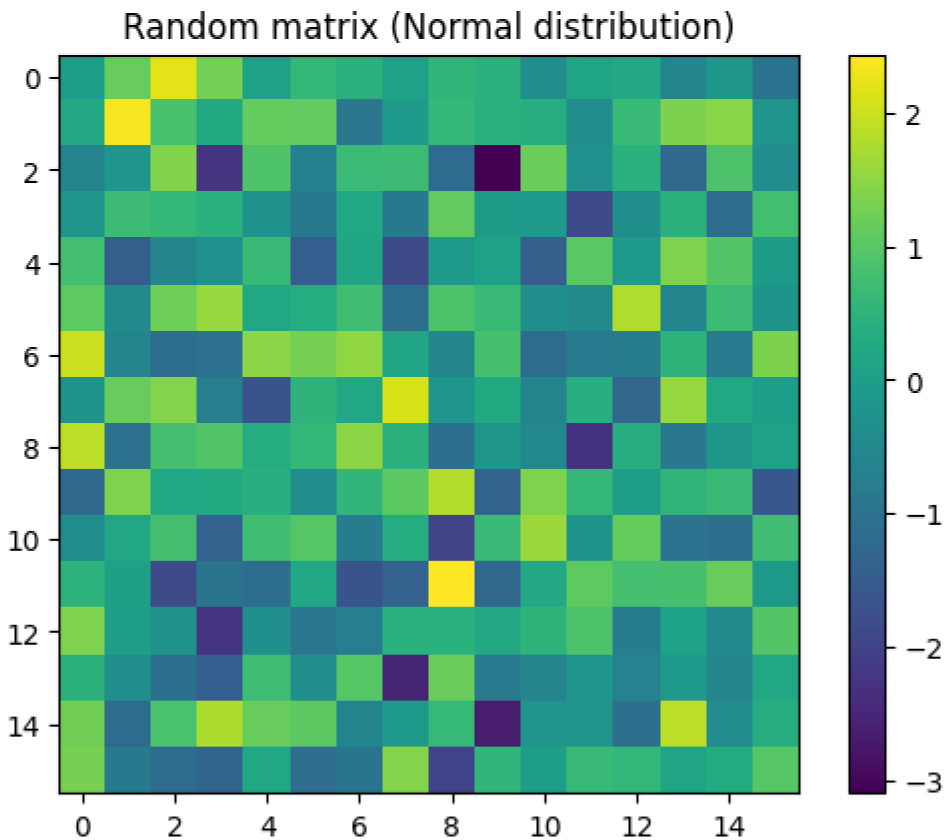


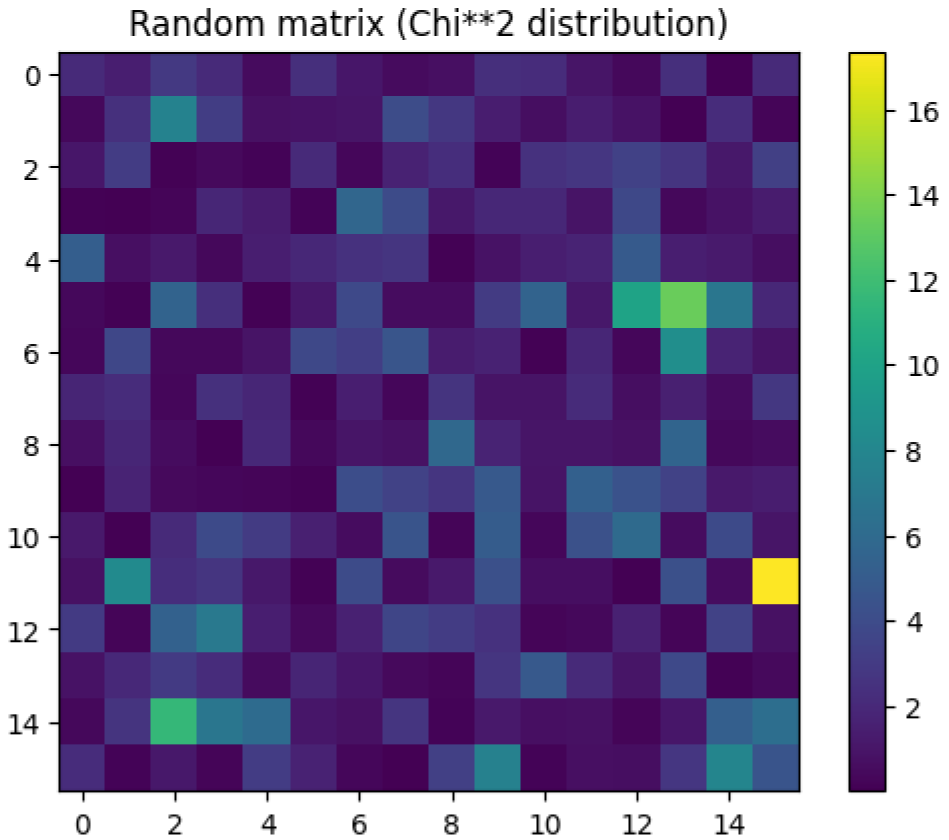
Note how element `(0, 0)` appears at the top-left of the figure.

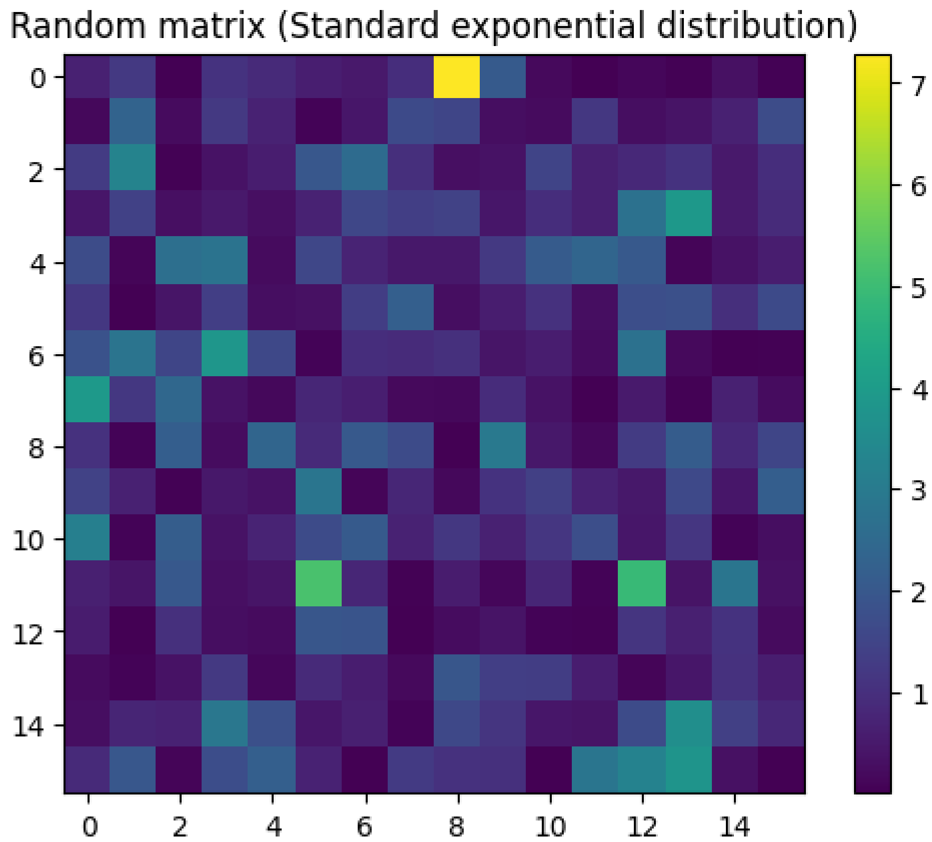
Challenge:
What are other distributions that you can generate? Check out the documentation for `np.random` and detail other random variables that you can generate through numpy and other packages.

In [47]:
```
# add code here

Z = np.random.normal(0, 1, (16, 16))
plt.imshow(Z)
plt.title('Random matrix (Normal distribution)')
plt.colorbar()
plt.show()

Z = np.random.chisquare(2, (16, 16))
```

```
plt.imshow(Z)
plt.title('Random matrix (Chi**2 distribution)')
plt.colorbar()
plt.show()

Z = np.random.standard_exponential((16, 16))
plt.imshow(Z)
plt.title('Random matrix (Standard exponential distribution)')
plt.colorbar()
plt.show()
```



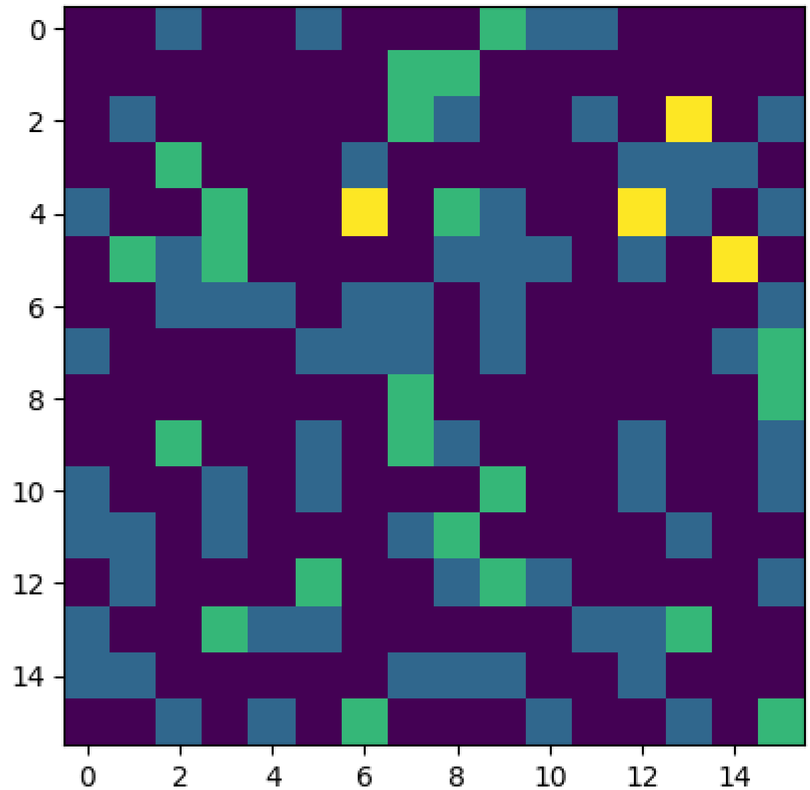

Random matrix (Chi**2 distribution)
0
2
4
6
8
10
12
14
16



4.5.2. Poisson distribution

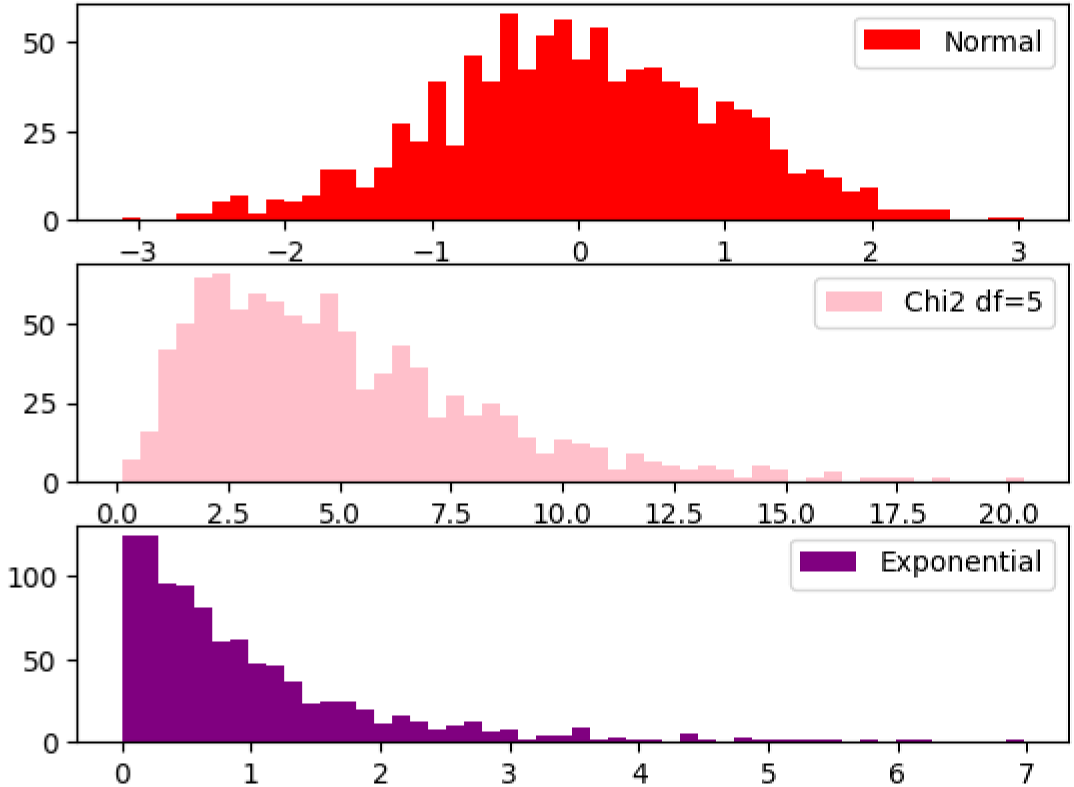
In [48]:
```
Z = np.random.poisson(0.5, (16,16))
plt.imshow(Z)
```

Out[48]:
```
<matplotlib.image.AxesImage at 0x7d520cb13460>
```



In [49]:
```
# Let's take a moment to plot the distribution as a histogram
# add code here:

fig, (ax1, ax2, ax3) = plt.subplots(3)

Normal_distr = np.random.normal(0,1,1000)
ax1.hist(Normal_distr, bins=50, label='Normal', color='r')
ax1.legend()

Chi2 = np.random.chisquare(5,1000)
ax2.hist(Chi2, bins=50, label='Chi2 df=5', color='pink')
ax2.legend()

Exp = np.random.standard_exponential(1000)
ax3.hist(Exp, bins=50, label='Exponential', color='purple')
ax3.legend()
```

Out[49]:
```
<matplotlib.legend.Legend at 0x7d520ca57cd0>
```

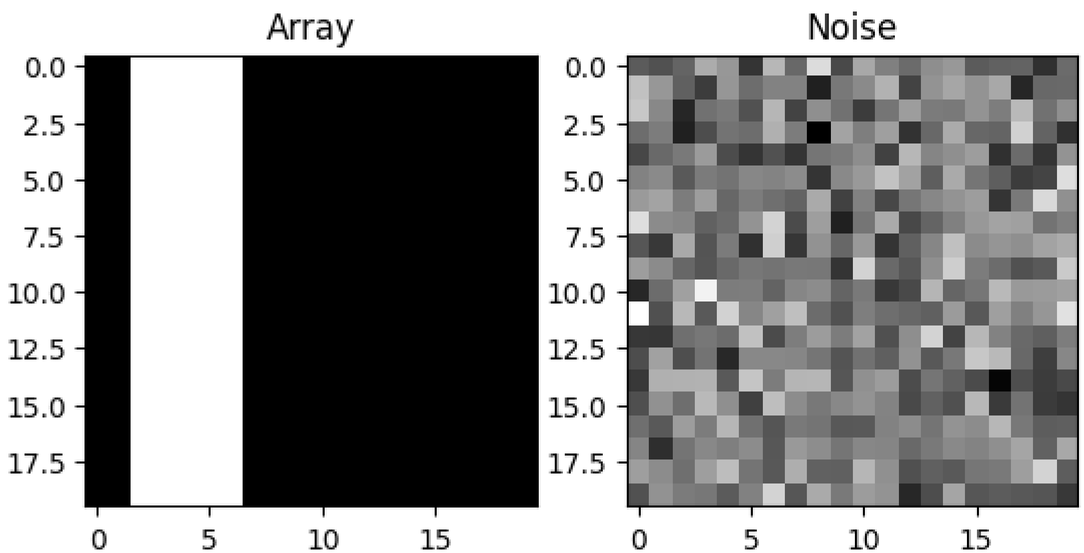


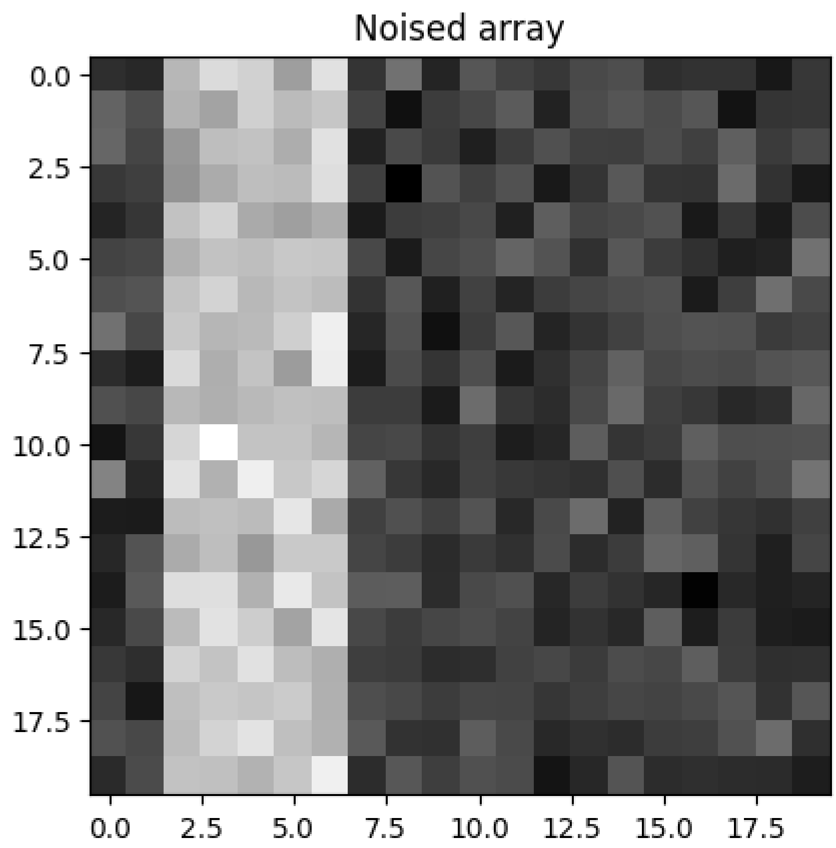
Challenge:

- Create a 20x20 array that has a white vertical bar (5 pixels wide) on a black background with additive Gaussian noise. Plot the output and include the code below.
- Create code to generate an image array (user specified size) that has rotated white bar (of a user specified angle), with additive poisson noise of user specified noise intensity parameter. Use this code to generate multiple example images and then plot them as subfigures.

In [50]:
```
# add code here

# Challenge 1
# 1. Array
matrix=np.zeros((20,20))
position_bar = np.random.randint(0,15)
matrix[:, position_bar:position_bar+5] = 255
```

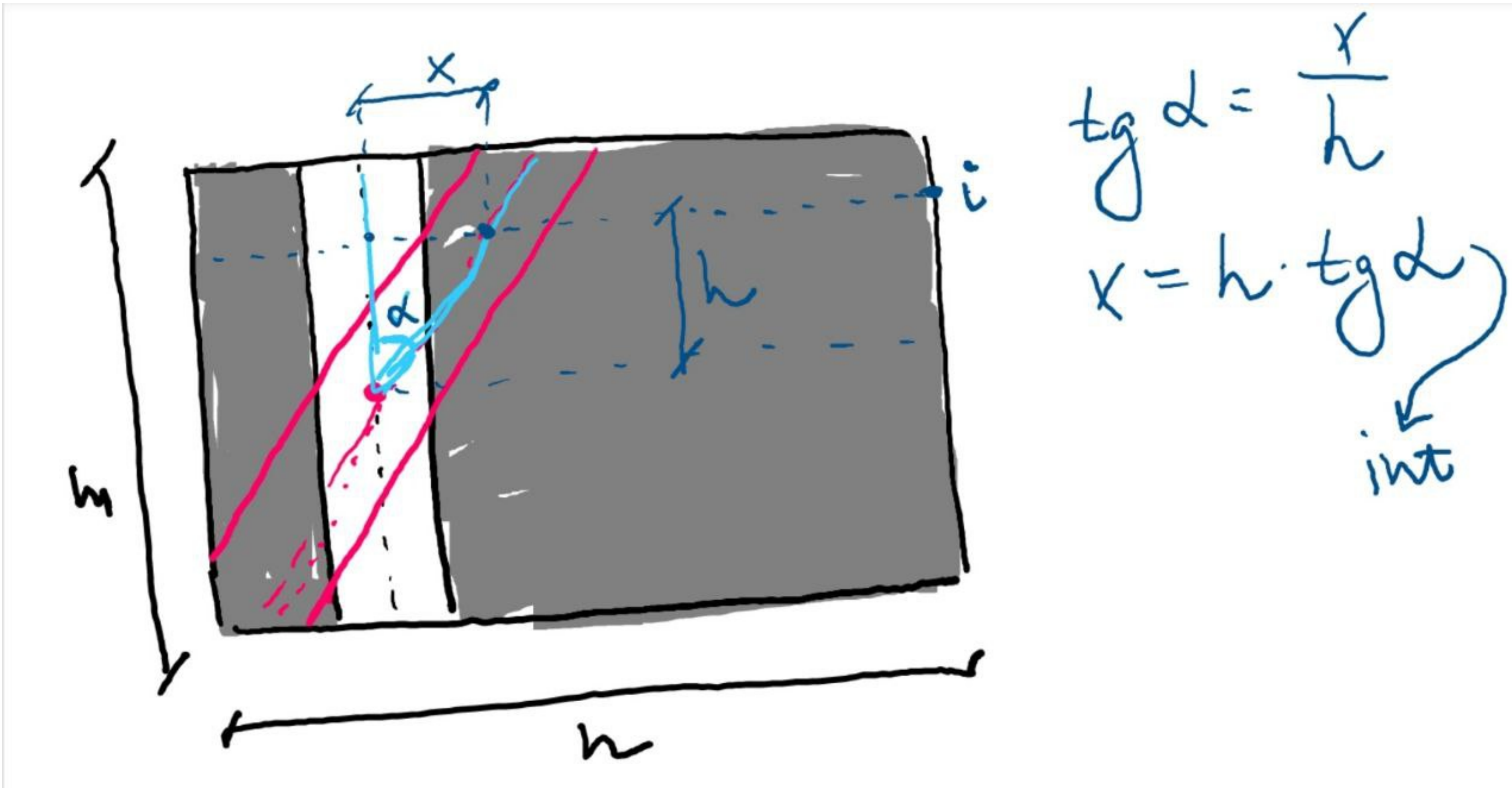
```
# 2. Noise
noise = (np.random.normal(0,1,(20,20)))
noise = (noise-noise.min())/(noise.max()-noise.min())*255

fig, (ax1, ax2) = plt.subplots(1,2)
ax1.imshow(matrix, cmap='gray', vmin=0, vmax=255)
ax1.set_title('Array')
ax2.imshow(noise, cmap='gray', vmin=0, vmax=255)
ax2.set_title('Noise')
plt.show()

# 3. Add
add_noise = matrix+noise
add_noise = (add_noise-add_noise.min())/(add_noise.max()-add_noise.min())*255
plt.imshow(add_noise, cmap='gray', vmin=0, vmax=255)
plt.title('Noised array')
plt.show()
```





In [51]:
```
# Challenge 2

def generate_noisy_array(m, n, angle, lambda_noise):
    # 1. Array
    matrix=np.zeros((m, n))
    position_bar = np.random.randint(0,min(m,n)-5) # random vertical bar position, because it's not specified in the ta
    mean_pos = m/2 # point of rotation
    for i in range(0,m):
        h = mean_pos-i
        x = round(math.tan(angle)*h) # pixels shift
        if (position_bar+x >= n) or (position_bar+x+5 <= 0):
            continue;
        matrix[i, max(0, (position_bar+x)):min(n, (position_bar+x+5))] = 255

    # 2. Noise
    pois_noise = np.random.poisson(lambda_noise, (m,n))
    pois_noise = (pois_noise-pois_noise.min())/(pois_noise.max()-pois_noise.min())*255
```

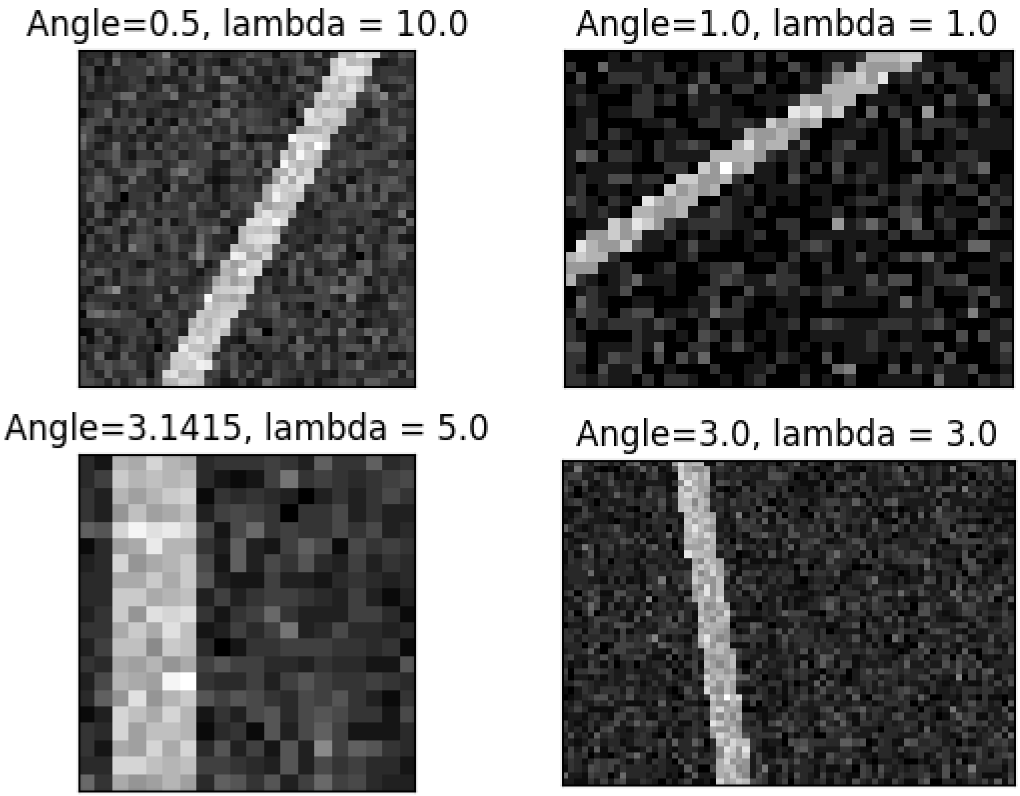
```
    # 3. Add noise
    add_noise = matrix+pois_noise
    add_noise = (add_noise-add_noise.min())/(add_noise.max()-add_noise.min())*255

    return add_noise
```



In [52]:
```
fig, axs = plt.subplots(2, 2)

for i in range(2):
  for j in range(2):
    print(i+j+1, 'Picture #')
    print('Specify array size (2 values, m and n):')
    m = int(input())
    n = int(input())

    print('Specify rotation angle (in radians)')
```

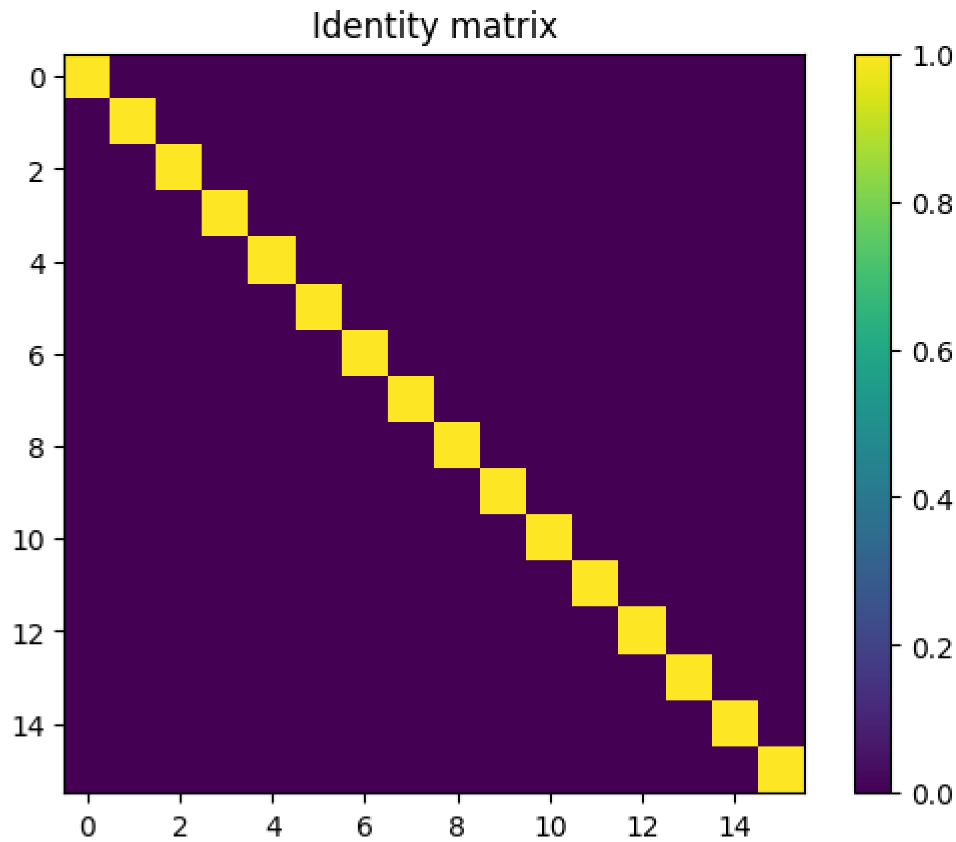
```
    angle = float(input())

    print('Specify noise intensity parameter:')
    lambda_noise = float(input())

    img = generate_noisy_array(m, n, angle, lambda_noise)
    axs[i,j].imshow(img, cmap='gray', vmin=0, vmax=255)
    axs[i,j].set_title(f'Angle={angle}, lambda = {lambda_noise}')
    axs[i,j].set_xticks([])
    axs[i,j].set_yticks([])
```

```
1 Picture #
Specify array size (2 values, m and n):
40
40
Specify rotation angle (in radians)
0.5
Specify noise intensity parameter:
10
2 Picture #
Specify array size (2 values, m and n):
30
40
Specify rotation angle (in radians)
1
Specify noise intensity parameter:
1
2 Picture #
Specify array size (2 values, m and n):
20
20
Specify rotation angle (in radians)
3.1415
Specify noise intensity parameter:
5
3 Picture #
Specify array size (2 values, m and n):
50
70
Specify rotation angle (in radians)
3
Specify noise intensity parameter:
3
```



4.5.3. Identity Matrix

In [53]:
```python
Z = np.eye(16)

plt.imshow(Z)
plt.title('Identity matrix')
plt.colorbar()
plt.show()
```



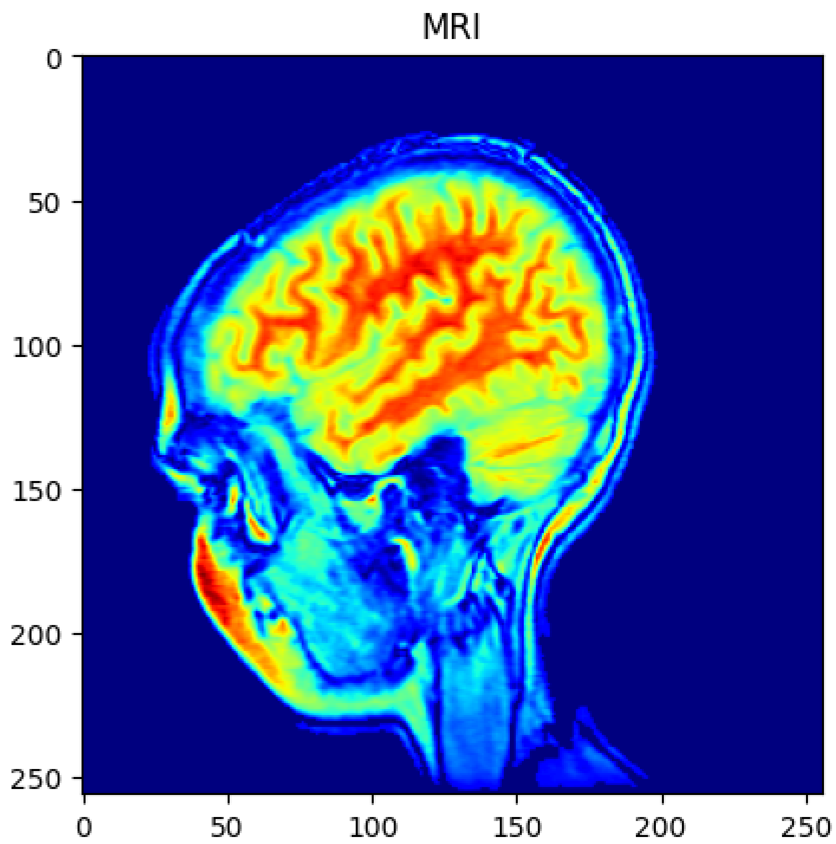
4.6. Visualizing images

In [54]:

```
# Loading a sample image
import matplotlib.cbook as cbook

w, h = 256, 256
with cbook.get_sample_data('s1045.ima.gz') as datafile:
    s = datafile.read()
img = np.frombuffer(s, np.uint16).astype(float).reshape((w, h))

print(img)
print('type:', img.dtype)
print('shape:', img.shape)
```

```
[[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.]]
type: float64
shape: (256, 256)
```

In [55]:
```
plt.imshow(img, cmap=plt.cm.jet)
plt.title('MRI')
plt.show()
```

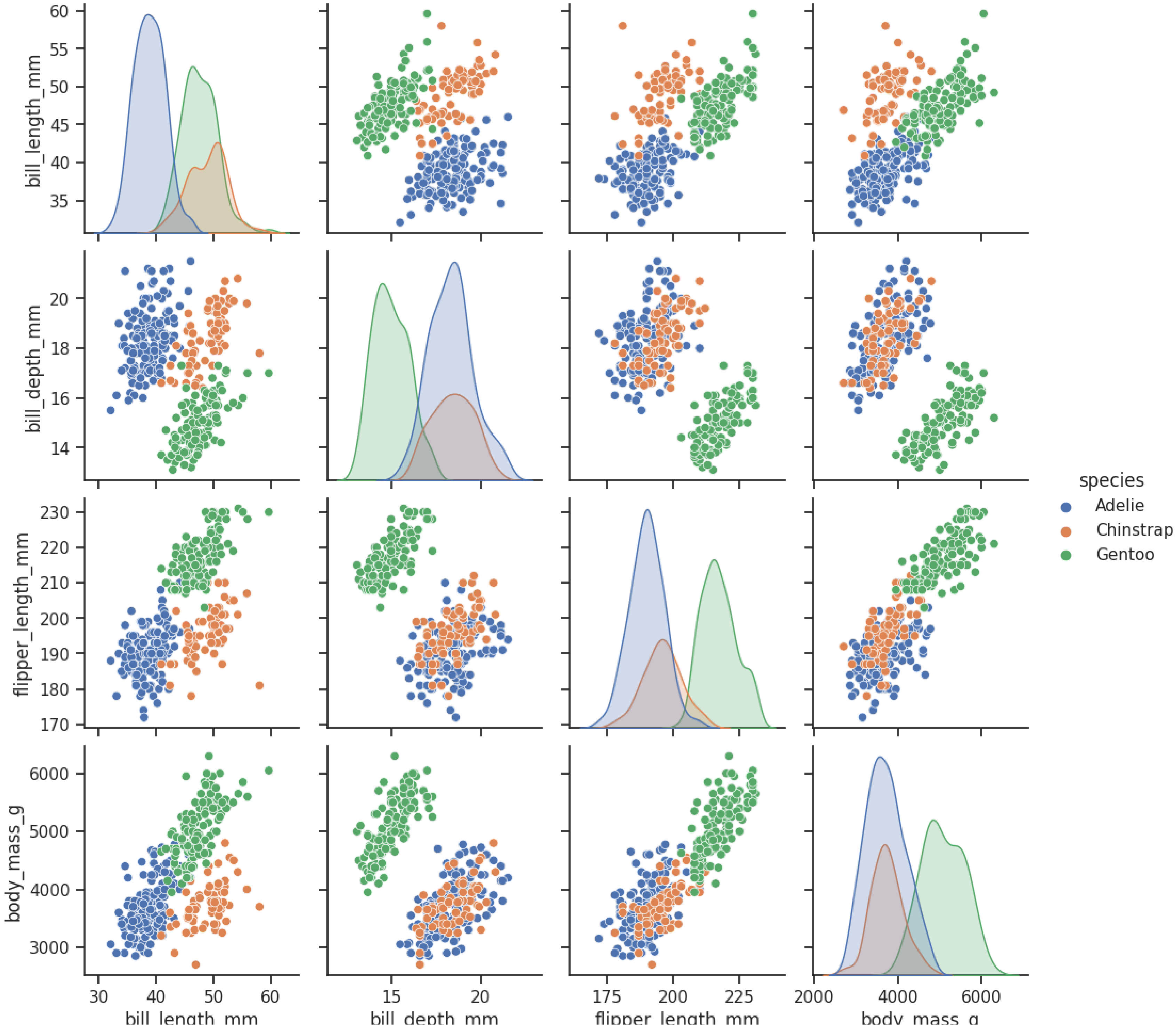


4.7. Visualizing dataframes using Seaborn

Seaborn provides a high-level interface for drawing attractive and informative statistical graphics. It is based on matplotlib.

Let's use it to visualize a widely used built in dataset consisting of different features and attributes collected from three different types of penguins (Adelie, Chinstrap, Gentoo).

In [56]:
```
import seaborn as sns
sns.set_theme(style="ticks")

df = sns.load_dataset("penguins", cache=False)
sns.pairplot(df, hue="species")
plt.show()
```


bill_length_mm
bill_depth_mm
flipper_length_mm
body_mass_g
species
Adelie
Chinstrap
Gentoo

bill_length_mm bill_depth_mm flipper_length_mm body_mass_g

In [57]:
```
df.head(10)
```

Out[57]:

	species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g	sex
0	Adelie	Torgersen	39.1	18.7	181.0	3750.0	Male
1	Adelie	Torgersen	39.5	17.4	186.0	3800.0	Female
2	Adelie	Torgersen	40.3	18.0	195.0	3250.0	Female
3	Adelie	Torgersen	NaN	NaN	NaN	NaN	NaN
4	Adelie	Torgersen	36.7	19.3	193.0	3450.0	Female
5	Adelie	Torgersen	39.3	20.6	190.0	3650.0	Male
6	Adelie	Torgersen	38.9	17.8	181.0	3625.0	Female
7	Adelie	Torgersen	39.2	19.6	195.0	4675.0	Male
8	Adelie	Torgersen	34.1	18.1	193.0	3475.0	NaN
9	Adelie	Torgersen	42.0	20.2	190.0	4250.0	NaN

4.8. More resources

- matplotlib cheatsheets: https://github.com/matplotlib/cheatsheets#cheatsheets
- matplotlib gallery: https://matplotlib.org/stable/gallery/index.html
- seaborn gallery: https://seaborn.pydata.org/examples/index.html

5. Pandas for table manipulation

pandas is a library for manipulating numerical tables and time series, and is a powerful tool for data analysis.

pandas dataframes are a very convenient way to interact with low-dimensional structured data. The basic dataframe object acts very similarly to an Excel file, but data can be manipulated with Python rather than clumsy Excel functions.

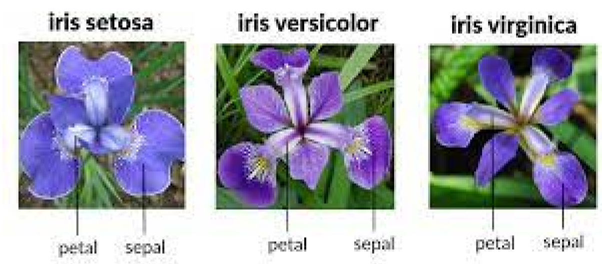
In [58]:
```
import pandas as pd
```

5.1. Loading data

Load in the iris dataset!

3 classes: Three different types of iris flowers, Iris Setosa, Iris Versicolor, and Iris Virginica.

4 features: Petal length and width, Sepal length and width



In [59]:
```
df = pd.read_csv("https://gist.githubusercontent.com/netj/8836201/raw/6f9306ad21398ea43cba4f7d537619d0e07d5ae3/iris.csv
```

In [60]:
```
df.head(20)
```

Out[60]:

	sepal.length	sepal.width	petal.length	petal.width	variety
0	5.1	3.5	1.4	0.2	Setosa
1	4.9	3.0	1.4	0.2	Setosa
2	4.7	3.2	1.3	0.2	Setosa
3	4.6	3.1	1.5	0.2	Setosa
4	5.0	3.6	1.4	0.2	Setosa
5	5.4	3.9	1.7	0.4	Setosa
6	4.6	3.4	1.4	0.3	Setosa
7	5.0	3.4	1.5	0.2	Setosa
8	4.4	2.9	1.4	0.2	Setosa
9	4.9	3.1	1.5	0.1	Setosa
10	5.4	3.7	1.5	0.2	Setosa
11	4.8	3.4	1.6	0.2	Setosa
12	4.8	3.0	1.4	0.1	Setosa
13	4.3	3.0	1.1	0.1	Setosa
14	5.8	4.0	1.2	0.2	Setosa
15	5.7	4.4	1.5	0.4	Setosa
16	5.4	3.9	1.3	0.4	Setosa
17	5.1	3.5	1.4	0.3	Setosa
18	5.7	3.8	1.7	0.3	Setosa
19	5.1	3.8	1.5	0.3	Setosa

5.2. Computing statistics

In [61]:

```
df['variety'].value_counts()
```

Out[61]:
```
Setosa        50
Versicolor    50
Virginica     50
Name: variety, dtype: int64
```

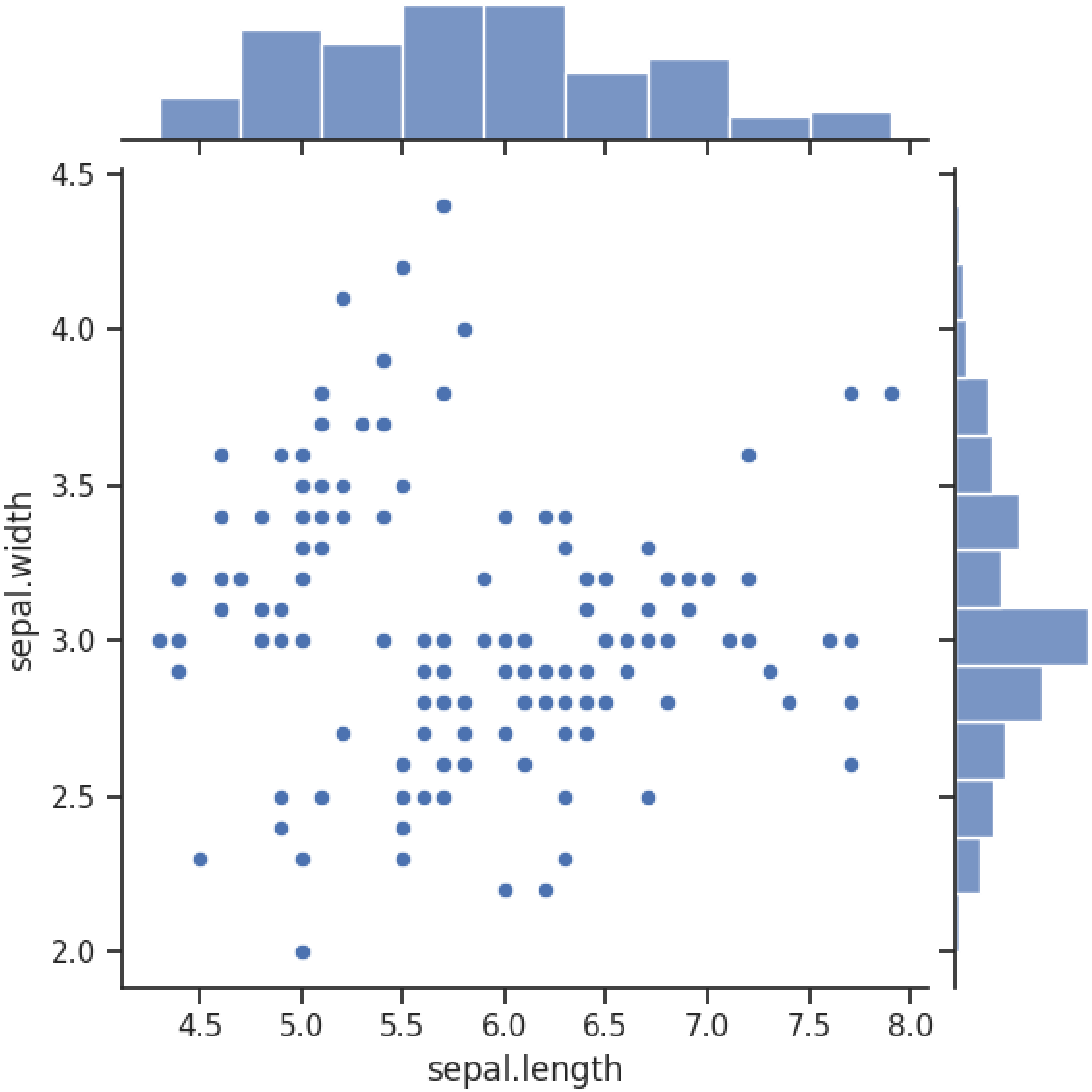
In [62]:
```
df.describe()
```

Out[62]:

	sepal.length	sepal.width	petal.length	petal.width
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.057333	3.758000	1.199333
std	0.828066	0.435866	1.765298	0.762238
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

5.3. More visualization

In [63]:
```
sns.jointplot(x='sepal.length', y='sepal.width', data=df, height=6)
plt.show()
```



5.4. Computing new features

Because Pandas is designed to work with NumPy, most NumPy functions will work on DataFrame objects.

In [64]:
```
df['length_diff'] = df['sepal.length'] - df['petal.length']
print(df.head())
```

```
   sepal.length  sepal.width  petal.length  petal.width variety  length_diff
0           5.1          3.5           1.4          0.2  Setosa          3.7
1           4.9          3.0           1.4          0.2  Setosa          3.5
2           4.7          3.2           1.3          0.2  Setosa          3.4
3           4.6          3.1           1.5          0.2  Setosa          3.1
4           5.0          3.6           1.4          0.2  Setosa          3.6
```

5.5. More resources

- pandas cheatsheet: https://pandas.pydata.org/Pandas_Cheat_Sheet.pdf

Challenge:

- Load a data spreadsheet (CSV) of your choice loaded into Colab. Visualize the spreadsheet entries and metadata with Pandas.
- Generate a pairplot across some set of features in your dataset.
- Load three images into Colab that represent your interests. Crop and rescale them to all be of equal size, and visualize them as subplots using matplotlib.

In [65]:
```
# Please add code here

df = sns.load_dataset("mpg", cache=False)
print(df.describe())
print('\n\nEntries\n\n')
df.head(10)
```

```
              mpg   cylinders  displacement  horsepower       weight  \
count  398.000000  398.000000    398.000000  392.000000   398.000000
mean    23.514573    5.454774    193.425879  104.469388  2970.424623
std      7.815984    1.701004    104.269838   38.491160   846.841774
min      9.000000    3.000000     68.000000   46.000000  1613.000000
25%     17.500000    4.000000    104.250000   75.000000  2223.750000
50%     23.000000    4.000000    148.500000   93.500000  2803.500000
75%     29.000000    8.000000    262.000000  126.000000  3608.000000
max     46.600000    8.000000    455.000000  230.000000  5140.000000

       acceleration  model_year
count    398.000000  398.000000
mean      15.568090   76.010050
std        2.757689    3.697627
min        8.000000   70.000000
25%       13.825000   73.000000
50%       15.500000   76.000000
75%       17.175000   79.000000
max       24.800000   82.000000


Entries
```

Out[65]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year	origin	name
0	18.0	8	307.0	130.0	3504	12.0	70	usa	chevrolet chevelle malibu
1	15.0	8	350.0	165.0	3693	11.5	70	usa	buick skylark 320
2	18.0	8	318.0	150.0	3436	11.0	70	usa	plymouth satellite
3	16.0	8	304.0	150.0	3433	12.0	70	usa	amc rebel sst
4	17.0	8	302.0	140.0	3449	10.5	70	usa	ford torino
5	15.0	8	429.0	198.0	4341	10.0	70	usa	ford galaxie 500
6	14.0	8	454.0	220.0	4354	9.0	70	usa	chevrolet impala
7	14.0	8	440.0	215.0	4312	8.5	70	usa	plymouth fury iii
8	14.0	8	455.0	225.0	4425	10.0	70	usa	pontiac catalina
9	15.0	8	390.0	190.0	3850	8.5	70	usa	amc ambassador dpl

In [66]:
```
df['origin'].value_counts()
```

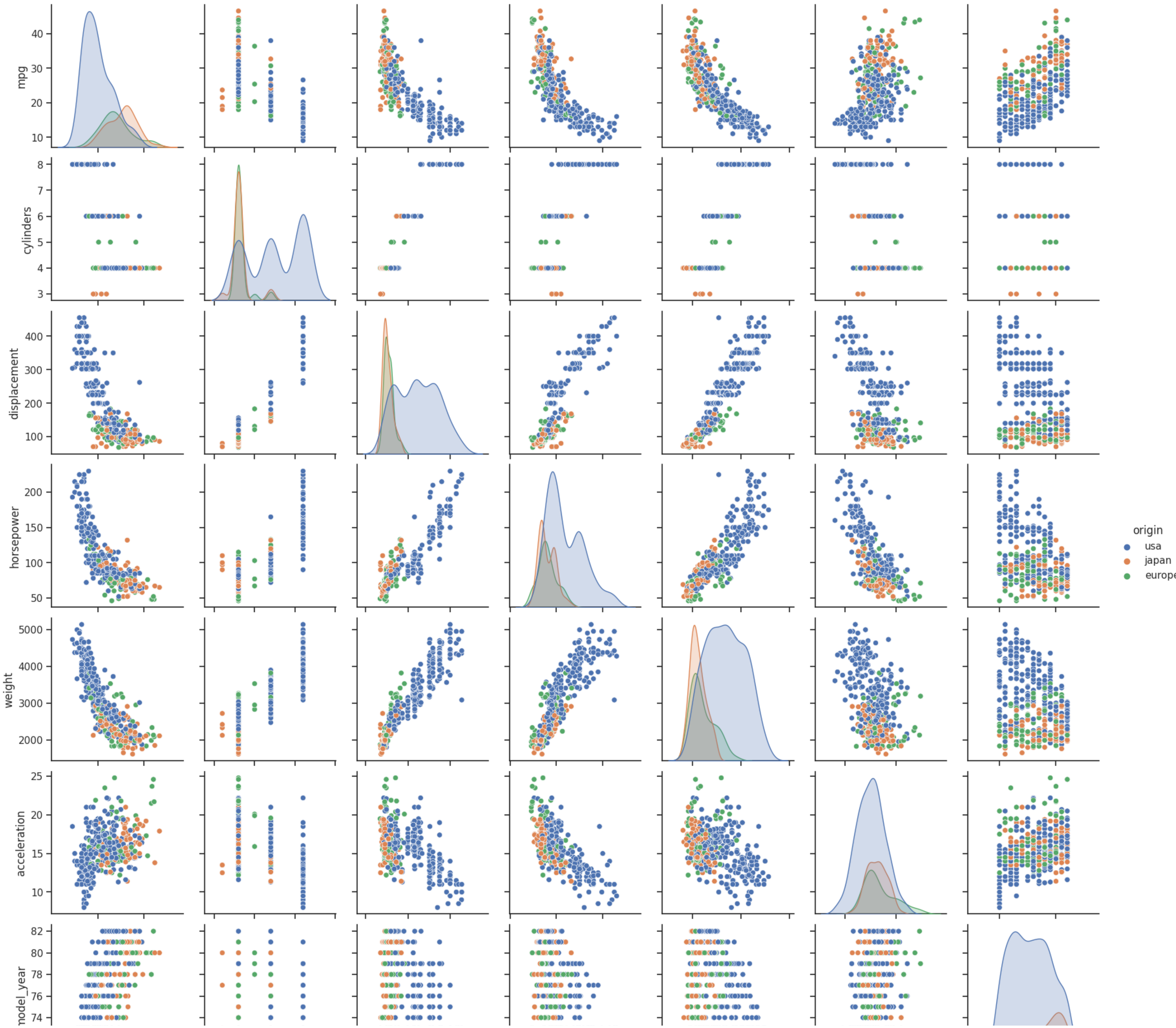
Out[66]:
```
usa       249
japan      79
europe     70
Name: origin, dtype: int64
```

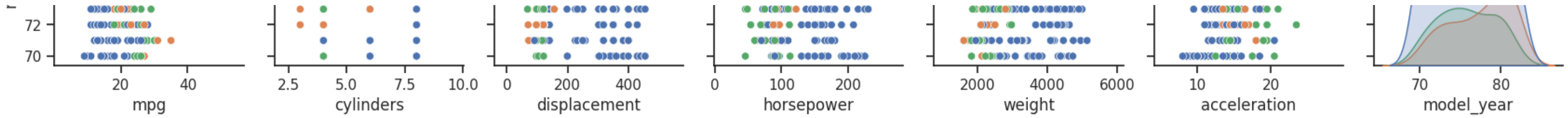
In [67]:
```
df['cylinders'].value_counts()
```

Out[67]:
```
4    204
8    103
6     84
3      4
5      3
Name: cylinders, dtype: int64
```

In [68]:
```
sns.pairplot(df, hue="origin")
```

Out[68]:
```
<seaborn.axisgrid.PairGrid at 0x7d51f31c8e80>
```


mpg
cylinders
displacement
horsepower
weight
acceleration
model_year
origin
usa
japan
europe



In [ ]:
```
# Challenge

# Load three images into Colab that represent your interests.
# Crop and rescale them to all be of equal size,
# and visualize them as subplots using matplotlib.

! wget 'https://github.com/kseniashilova/BMED6517/blob/main/brain.jpg?raw=true'
! wget 'https://github.com/kseniashilova/BMED6517/blob/main/math.jpeg?raw=true'
! wget 'https://github.com/kseniashilova/BMED6517/blob/main/programming.jpg?raw=true'
```

In [70]:
```
import matplotlib.pyplot as plt
import matplotlib.image as img

brain_img = img.imread('brain.jpg?raw=true')
math_img = img.imread('math.jpeg?raw=true')
programming_img = img.imread('programming.jpg?raw=true')
```

In [71]:
```
print('shapes: ', brain_img.shape, math_img.shape, programming_img.shape)
print('ratio: ', brain_img.shape[0]/brain_img.shape[1],
      math_img.shape[0]/math_img.shape[1],
      programming_img.shape[0]/programming_img.shape[1])
overall_ration=min(brain_img.shape[0]/brain_img.shape[1],
      math_img.shape[0]/math_img.shape[1],
      programming_img.shape[0]/programming_img.shape[1])

brain_size=[int(brain_img.shape[1]*overall_ration), brain_img.shape[1], 3]
math_size=[int(math_img.shape[1]*overall_ration), math_img.shape[1], 3]
programming_size=[int(programming_img.shape[1]*overall_ration), programming_img.shape[1], 3]
```

```
shapes:  (360, 540, 3) (288, 512, 3) (630, 1200, 3)
ratio:  0.6666666666666666 0.5625 0.525
```

In [72]:
```
# crop
print('new shapes: ', brain_size, math_size, programming_size)
brain_img = brain_img[:brain_size[0], :, :]
math_img = math_img[:math_size[0], :, :]
programming_img = programming_img[:programming_size[0], :, :]
```

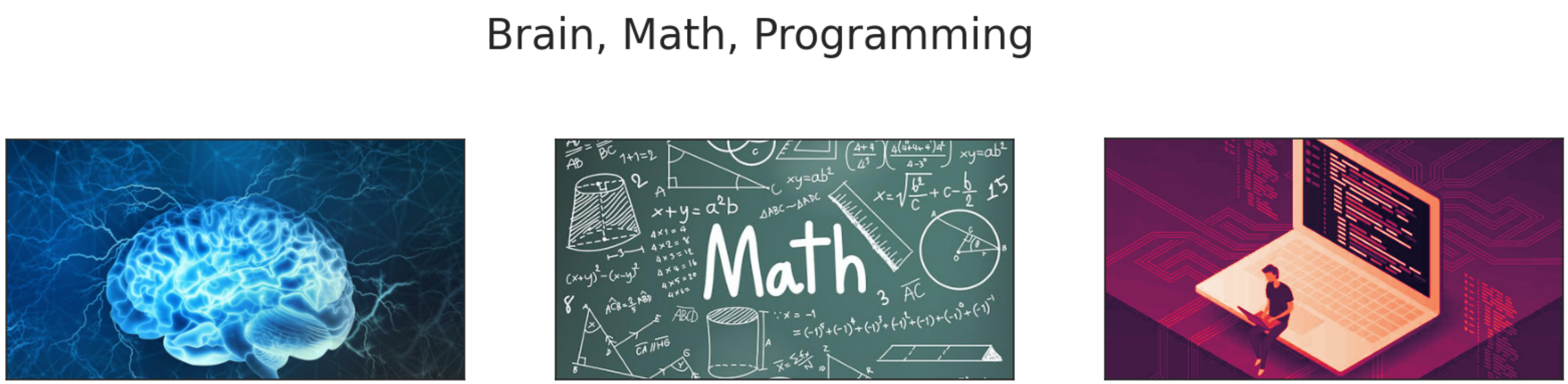
```
new shapes:  [283, 540, 3] [268, 512, 3] [630, 1200, 3]
```

In [73]:
```
# rescale
from skimage.transform import rescale

new_width = min(brain_img.shape[0], math_img.shape[0], programming_img.shape[0])
print('new_width: ', new_width)

new_brain = rescale(brain_img, new_width/brain_img.shape[0], anti_aliasing=False)
new_math = rescale(math_img, new_width/math_img.shape[0], anti_aliasing=False)
new_programming = rescale(programming_img, new_width/programming_img.shape[0],
                          anti_aliasing=False)
```

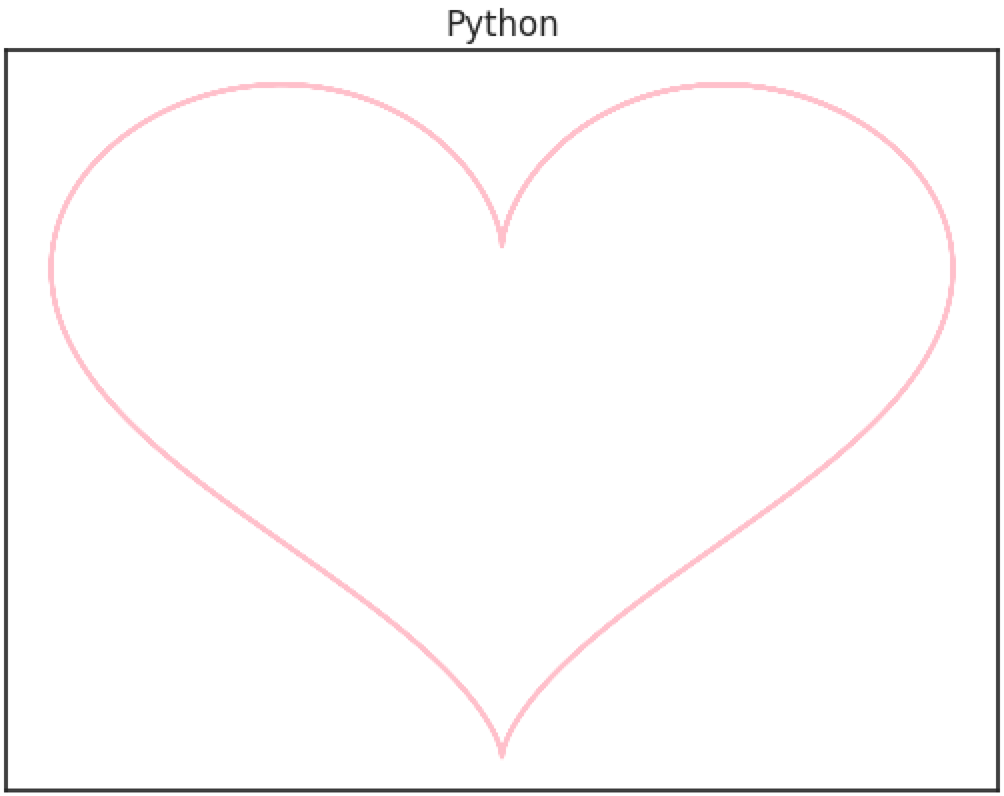
```
new_width:  268
```

In [74]:
```
fig, axs = plt.subplots(1, 3, figsize=(20, 5))
axs[0].imshow(new_brain)
axs[1].imshow(new_math)
axs[2].imshow(new_programming)
for i in range(3):
    axs[i].set_xticks([])
    axs[i].set_yticks([])
plt.suptitle('Brain, Math, Programming', fontsize=30)
```

Out[74]:
```
Text(0.5, 0.98, 'Brain, Math, Programming')
```



In [75]:
```
t = np.linspace(0,20,1000)
x = 16*np.sin(t)**3
y = 13*np.cos(t)-5*np.cos(2*t)-2*np.cos(3*t)-np.cos(4*t)
plt.plot(x, y, 'pink')
plt.xticks([])
plt.yticks([])
plt.title('Python')
```



Contributors: Mehdi Azabou (mazabou{at}gatech.edu), Eva Dyer (evadyer{at}gatech.edu)