Python Fundamentals

Dr. Ram Prasad K
VisionCog R&D

ram.krish@visioncog.com
https://www.visioncog.com

Python Fundamentals

Containers

List

```
# Create a list
list1 = [3, 1, 2]

print(list1, list1[2])
# [3, 1, 2] 2

# Negative indices count from
# the end of the list;
print(list1[-1])
# 2

# Lists can contain elements
# of different types
list1[2] = 'foo'
print(list1)
# [3, 1, 'foo']

# Add a new element to
# the end of the list
list1.append('bar')

print(list1)
# [3, 1, 'foo', 'bar']

# Remove and return the
# last element of the list
list2 = list1.pop()

print(list2, list1)
# bar [3, 1, 'foo']
```

Python Fundamentals

Containers

List slicing

```
                0       1       2       3       4
animals = ['cat', 'dog', 'monkey', 'tiger', 'lion']

print(animals)
# ['cat', 'dog', 'monkey', 'tiger', 'lion']


# Get a slice from index 2 to 4 (exclusive);
print(animals[2:4])
# ['monkey', 'tiger']


# Get a slice from index 2 to the end;
print(animals[2:])
# ['monkey', 'tiger', 'lion']


# Get a slice from the start to index 2 (exclusive);
print(animals[:2])
# ['cat', 'dog']
```

Python Fundamentals

Containers

List slicing

```
# Get a slice of the whole list;
print(animals[:])
# ['cat', 'dog', 'monkey', 'tiger', 'lion']


# Slice indices can be negative;
print(animals[:-1])
# ['cat', 'dog', 'monkey', 'tiger']


# Assign a new sublist to a slice
animals[2:4] = ['deer', 'horse']

print(animals)
# ['cat', 'dog', 'deer', 'horse', 'lion']
```

Python Fundamentals

Containers

List loops

```
animals = ['cat', 'dog', 'monkey']

for elems in animals:
    print(elems)

# cat
# dog
# monkey
```

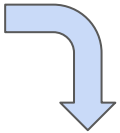
To access only the elements

Python Fundamentals

Containers

List loops

```
animals = ['cat', 'dog', 'monkey']

for idx, elem in enumerate(animals):
    print('%d: %s' % (idx, elem))

# 0: cat
# 1: dog
# 2: monkey
```



To access also index within 'for' loop

Python Fundamentals

Containers

List comprehension

```
nums = [0, 1, 2, 3, 4]
squares = []

for x in nums:
    squares.append(x ** 2)

print(squares)
# Prints [0, 1, 4, 9, 16]
```

```
nums = [0, 1, 2, 3, 4]

squares = [x ** 2 for x in nums]
print(squares)
# Prints [0, 1, 4, 9, 16]
```

```
nums = [0, 1, 2, 3, 4]

even_squares = [x ** 2 for x in nums if x % 2 == 0]
print(even_squares)
# Prints "[0, 4, 16]"
```

List comprehensions can also contain conditions

Python Fundamentals

Containers

Dictionary

```
d = {'cat': 'cute', 'dog': 'furry'}  # Create a new dictionary with some data

print(d['cat'])
# Get an entry from a dictionary; prints "cute"

print('cat' in d)
# Check if a dictionary has a given key; prints "True"

d['fish'] = 'wet'
# Set an entry in a dictionary

print(d['fish'])
# Prints "wet"

# print(d['monkey'])  # KeyError: 'monkey' not a key of d
```

Dictionaries are like associative arrays

Index can be strings, unlike list

Python Fundamentals

Containers

Dictionary loops

```
d = {'person': 2, 'cat': 4, 'spider': 8}

for animal in d:
    legs = d[animal]
    print('A %s has %d legs' % (animal, legs))

# A person has 2 legs
# A cat has 4 legs
# A spider has 8 legs
```

iterating only the keys

Python Fundamentals

Containers

Dictionary loops

```
d = {'person': 2, 'cat': 4, 'spider': 8}

for animal, legs in d.items():
    print('A %s has %d legs' % (animal, legs))

# A person has 2 legs
# A cat has 4 legs
# A spider has 8 legs
```

iterating both key and its value

Python Fundamentals

Containers

Dictionary comprehension

```
nums = [0, 1, 2, 3, 4]

even_num_to_square = {x: x ** 2 for x in nums if x % 2 == 0}

print(even_num_to_square)
# {0: 0, 2: 4, 4: 16}
```

Python Fundamentals

Tuples

```
# Create a tuple
t = (8, 10, 25)

print(type(t))
# <class 'tuple'>

print(t[0], t[1], t[2])
# 8 10 25

for i in range(len(t)):
    print(t[i])

# 8
# 10
# 25
```

NUMPY

Dr. Ram Prasad K
VisionCog R&D

ram.krish@visioncog.com
https://www.visioncog.com

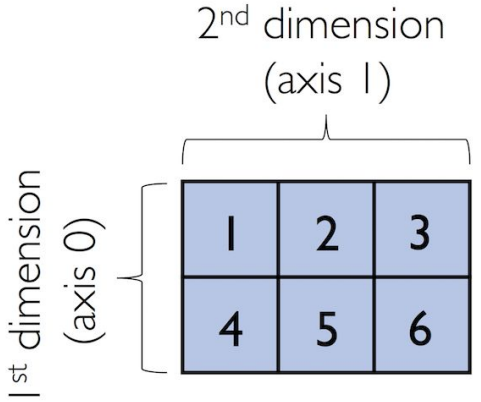
NUMPY

N-dimensional array

```
import numpy as np

a = np.array([1, 2, 3])   # Create a rank 1 array

print(type(a))            # Prints "<class 'numpy.ndarray'>"
print(a.shape)            # Prints "(3,)"

print(a[0], a[1], a[2])   # Prints "1 2 3"

a[0] = 5                  # Change an element of the array
print(a)                  # Prints "[5, 2, 3]"


b = np.array([[1,2,3],[4,5,6]])    # Create a rank 2 array

print(b.shape)                     # Prints "(2, 3)"
print(b[0, 0], b[0, 1], b[1, 0])   # Prints "1 2 4"
```

High performance multi-dimensional array data structures

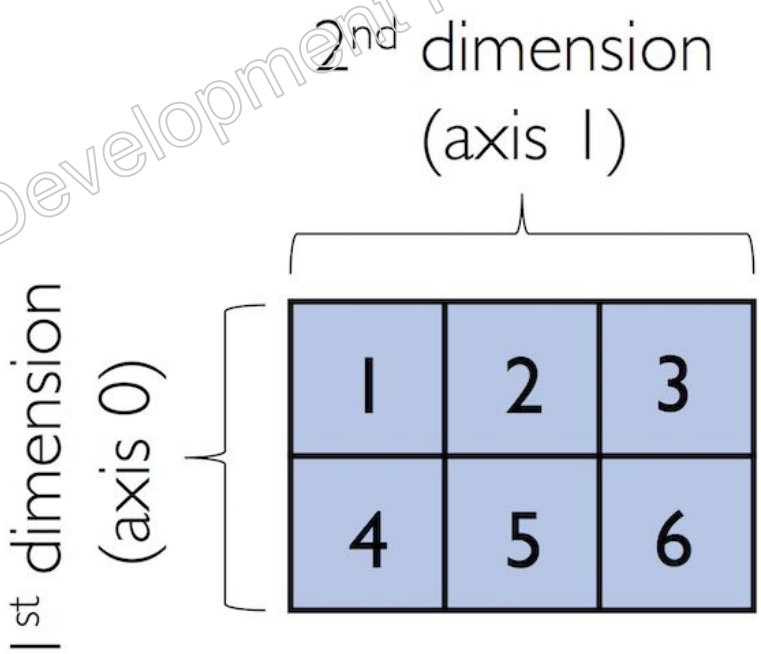
Mostly implemented in C language



NUMPY

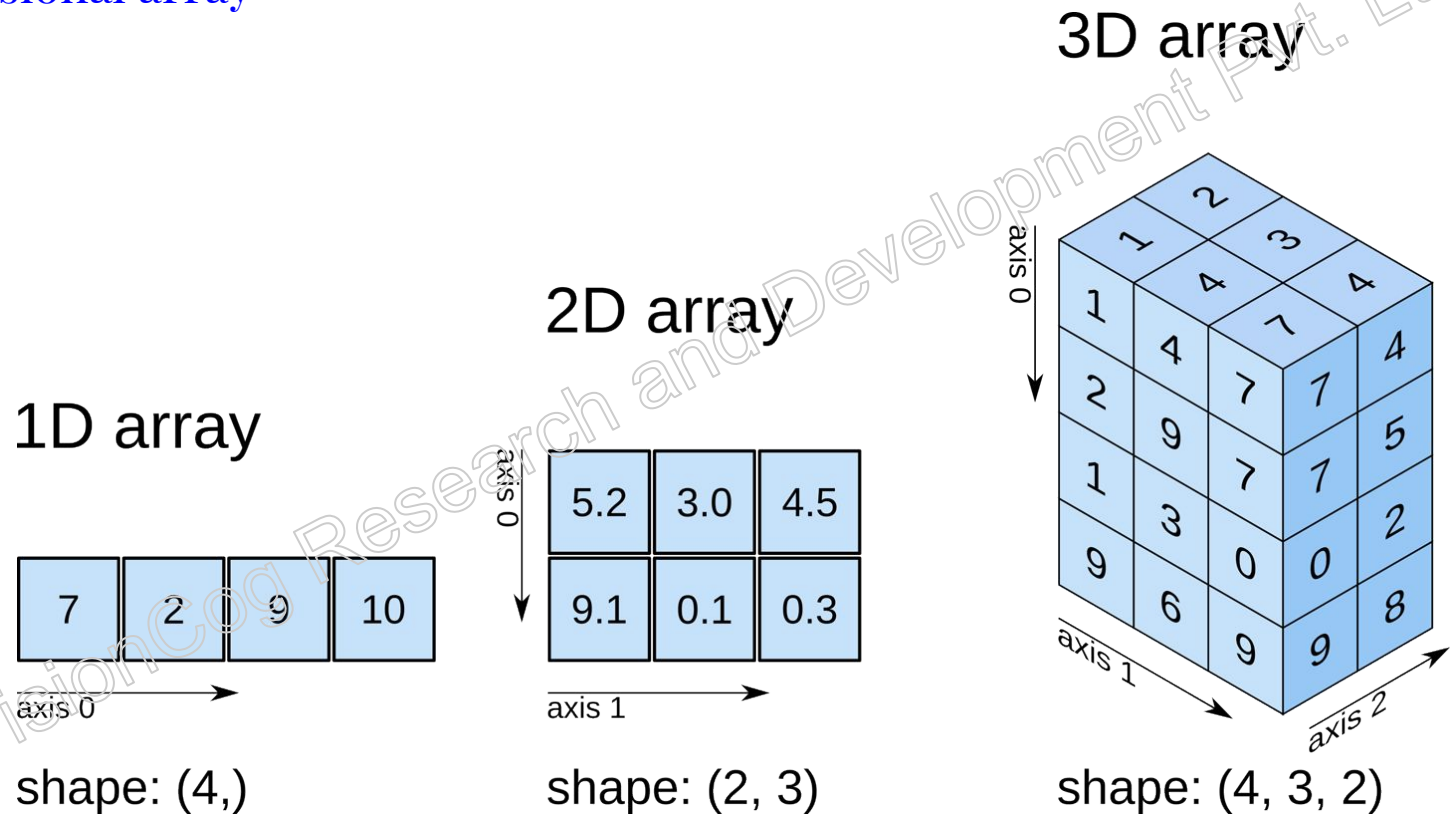
N-dimensional array

```
l = [[1, 2, 3], [4, 5, 6]]  # list of lists
ary2d = np.array(l)         # Convering list to array

print(ary2d)
# [[1 2 3]
#  [4 5 6]]

print(ary2d.dtype)
# Prints "int64"

float32_ary = ary2d.astype(np.float32)
# Converting the type of array

print(float32_ary)
# Prints "[[1. 2. 3.]
#         [4. 5. 6.]]"

print(ary2d.shape)
# Prints "(2,3)"
```



NUMPY

N-dimensional array



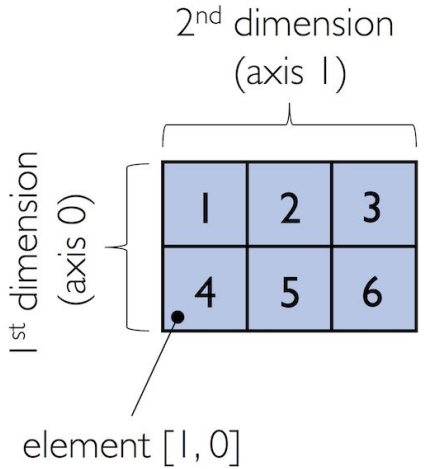
NUMPY

Array slicing

```
import numpy as np

# Create the following rank 2 array with shape (3, 4)
# [[ 1  2  3  4]
#  [ 5  6  7  8]
#  [ 9 10 11 12]]
a = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])

# Use slicing to pull out the subarray consisting of the first 2 rows
# and columns 1 and 2; b is the following array of shape (2, 2):
# [[2 3]
#  [6 7]]
b = a[:2, 1:3]

# A slice of an array is a view into the same data, so modifying it
# will modify the original array.
print(a[0, 1])   # Prints "2"

b[0, 0] = 77     # b[0, 0] is the same piece of data as a[0, 1]

print(a[0, 1])   # Prints "77"
```

Sliced result is actually a pointer to the original array.

Modifying the sliced result will modify the original array.



NUMPY

Array slicing

```
# Slicing is a pointer to the original array.
# Any modification applied to 'b' will be reflected in 'a'
# If we want a copy of the subarray, then use copy()

bc = a[:2, 1:3].copy() # also np.copy(a[:2, 1:3])

print(bc)
# [[77  3]
#  [ 6  7]]


# Now, modifying the contents of 'bc' will not affect 'a'
bc[0,0] = 88

print(bc)
# [[88  3]
#  [ 6  7]]

print(a)
# [[ 1 77  3  4]
#  [ 5  6  7  8]
#  [ 9 10 11 12]]
```

Sliced result is actually a pointer to the original array.

Modifying the sliced result will modify the original array.

Numpy

Array maths

```
import numpy as np

x = np.array([[1,2],[3,4]], dtype=np.float64)
y = np.array([[5,6],[7,8]], dtype=np.float64)

# Elementwise sum; both produce the array
# [[ 6.0  8.0]
#  [10.0 12.0]]
print(x + y)
print(np.add(x, y))

# Elementwise difference; both produce the array
# [[-4.0 -4.0]
#  [-4.0 -4.0]]
print(x - y)
print(np.subtract(x, y))
```

These operations are known as vectorized operations.

NUMPY

Array maths

```
# Elementwise product; both produce the array
# [[ 5.0 12.0]
#  [21.0 32.0]]
print(x * y)
print(np.multiply(x, y))

# Elementwise division; both produce the array
# [[ 0.2         0.33333333]
#  [ 0.42857143  0.5       ]]
print(x / y)
print(np.divide(x, y))

# Elementwise square root; produces the array
# [[ 1.          1.41421356]
#  [ 1.73205081  2.        ]]
print(np.sqrt(x))
```

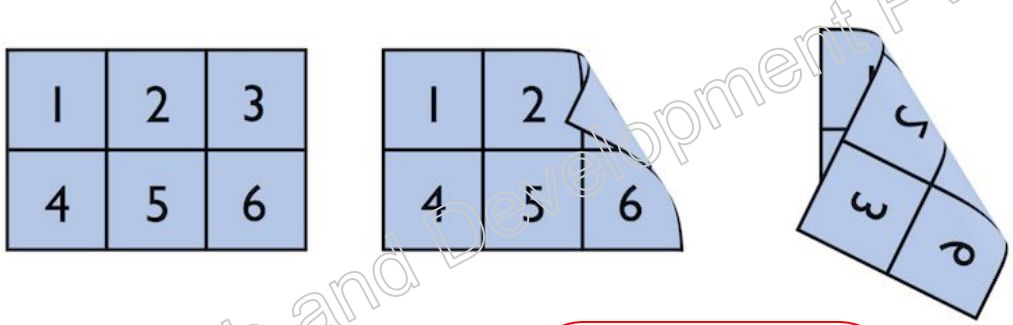
```
# To obtain matrix multiplication,
# use '@' or np.matmul()
# [[19. 22.]
#  [43. 50.]]
print(x @ y)
print(np.matmul(x,y))
```

These operations are known as vectorized operations.

NUMPY

Array transpose

```
import numpy as np

x = np.array([[1,2,3], [4,5,6]])

print(x)    # Prints "[[1 2 3]
            #          [4 5 6]]"

print(x.T)  # Prints "[[1 4]
            #          [2 5]
            #          [3 6]]"

# Note that taking the transpose of a rank 1 array does nothing:
v = np.array([1,2,3])

print(v)    # Prints "[1 2 3]"

print(v.T)  # Prints "[1 2 3]"
```

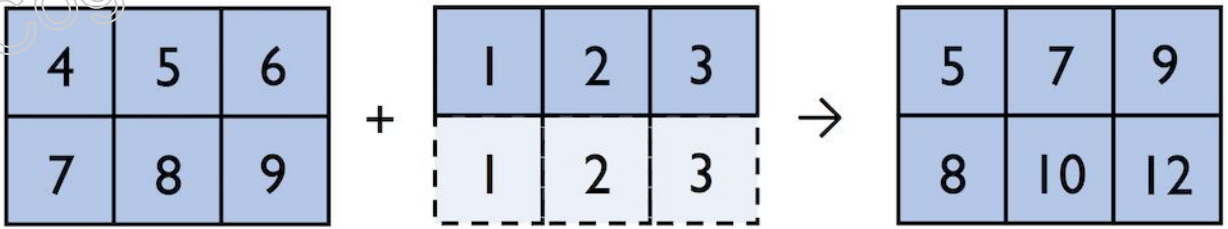



```
v1 = v.reshape((1,3))

print(v1)
# [[1 2 3]]

print(v1.shape)
# (1, 3)

print(v1.T)
# [[1]
#  [2]
#  [3]]
```

NUMPY

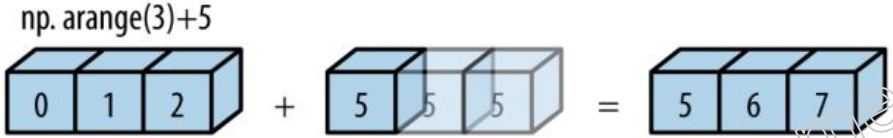
Array broadcasting

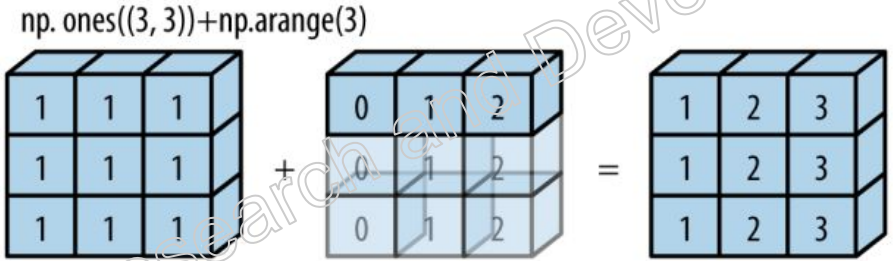
```
import numpy as np

# We will add the vector v to each row of the matrix x,
# storing the result in the matrix y
x = np.array([[4,5,6], [7,8,9]])

v = np.array([1, 2, 3])

y = x + v  # Add v to each row of x using broadcasting

print(y)  # Prints "[[ 5   7   9]
          #          [ 8  10  12]]"
```

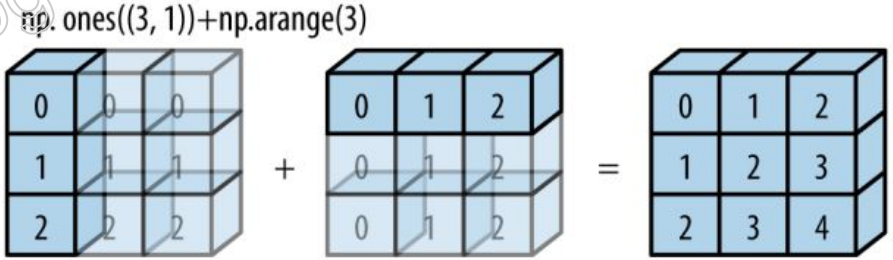


NUMPY

Array broadcasting







Matplotlib

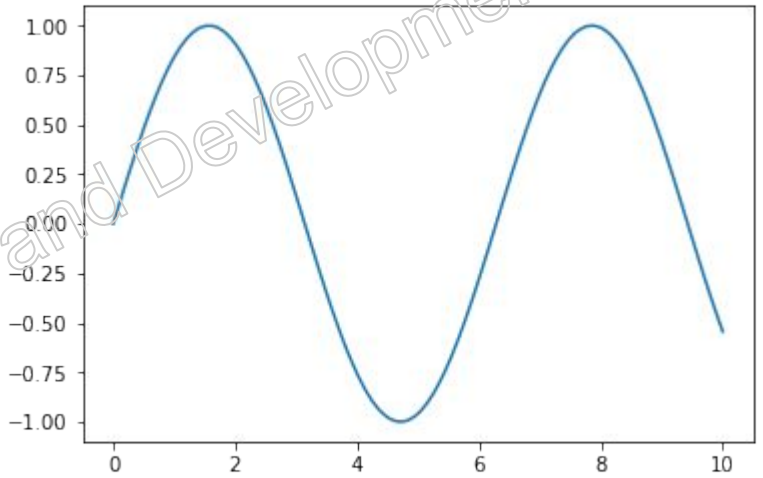
Dr. Ram Prasad K
VisionCog R&D

ram.krish@visioncog.com
https://www.visioncog.com

Matplotlib

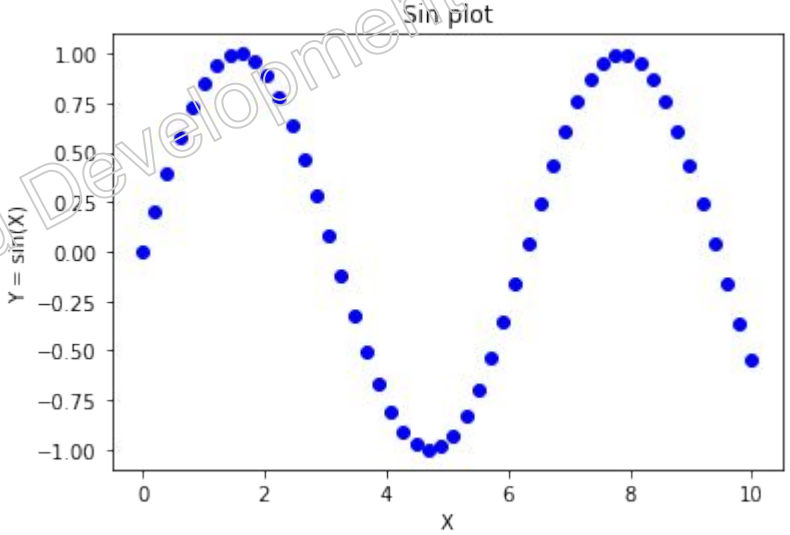
Matplotlib

```
%matplotlib inline
import matplotlib.pyplot as plt

x = np.linspace(0, 10, 100)
plt.plot(x, np.sin(x))
plt.show()
```



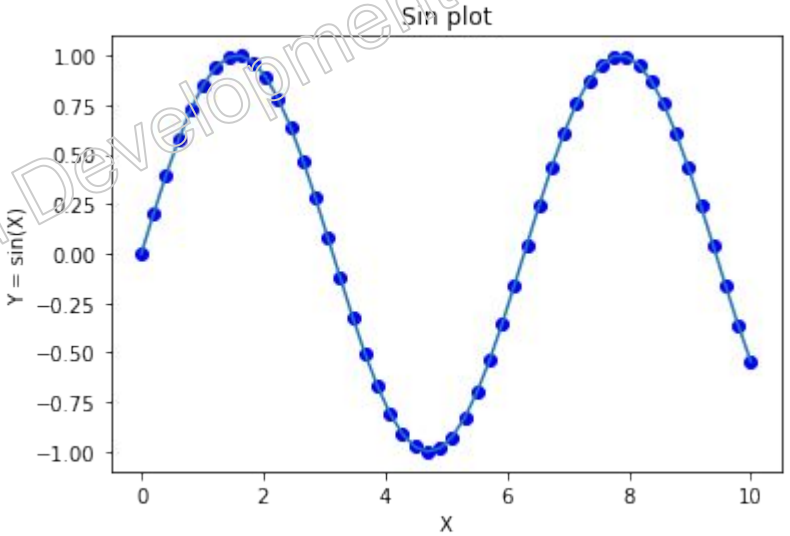
Matplotlib

```
X = np.linspace(0,10,50)
Y = np.sin(X)

plt.title("Sin plot")
plt.xlabel("X")
plt.ylabel("Y = sin(X)")

plt.plot(X, Y, 'o', color='blue')

plt.show()
```



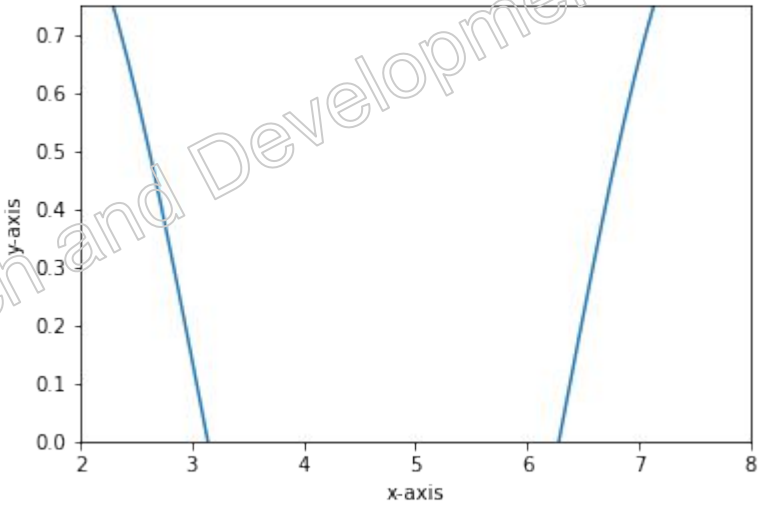
Matplotlib

```
X = np.linspace(0,10,50)
Y = np.sin(X)

plt.title("Sin plot")
plt.xlabel("X")
plt.ylabel("Y = sin(X)")

plt.plot(X, Y, 'o', color='blue')

plt.plot(X, Y)

plt.show()
```



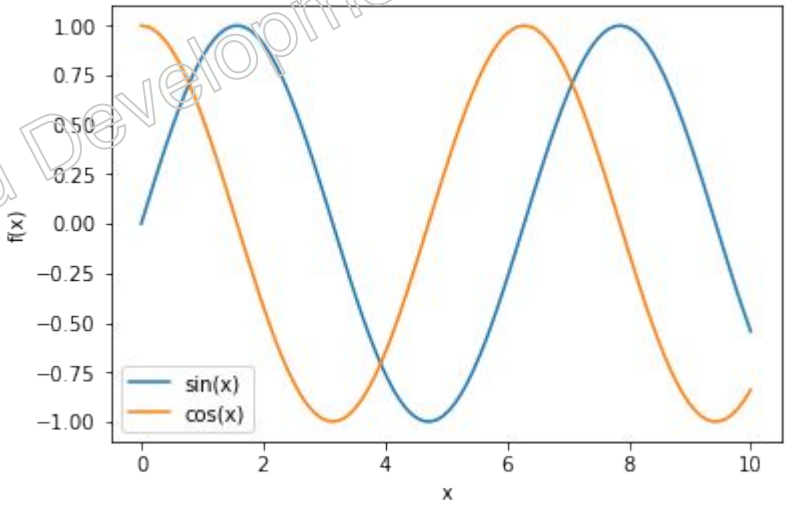
Matplotlib

```
x = np.linspace(0, 10, 100)
plt.plot(x, np.sin(x))

plt.xlim([2, 8])
plt.ylim([0, 0.75])

plt.xlabel('x-axis')
plt.ylabel('y-axis')

plt.show()
```



Matplotlib

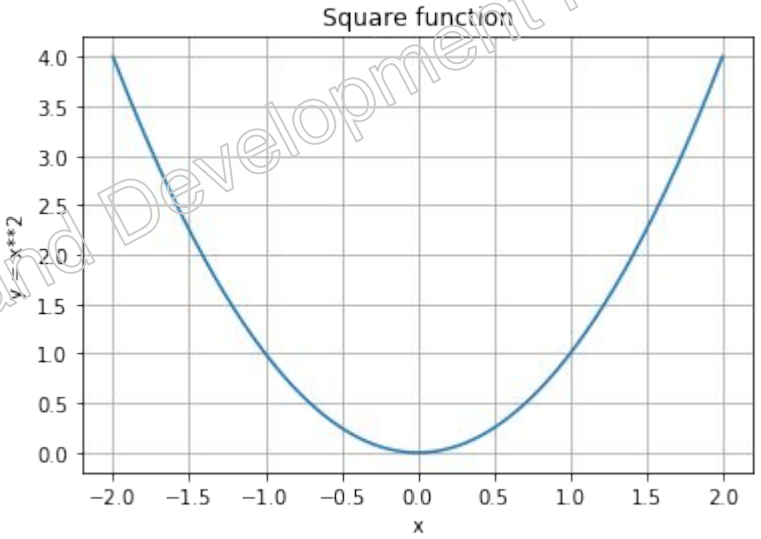
```
x = np.linspace(0, 10, 100)

plt.plot(x, np.sin(x), label=('sin(x)'))
plt.plot(x, np.cos(x), label=('cos(x)'))

plt.ylabel('f(x)')
plt.xlabel('x')

plt.legend(loc='lower left')
plt.show()
```



Matplotlib

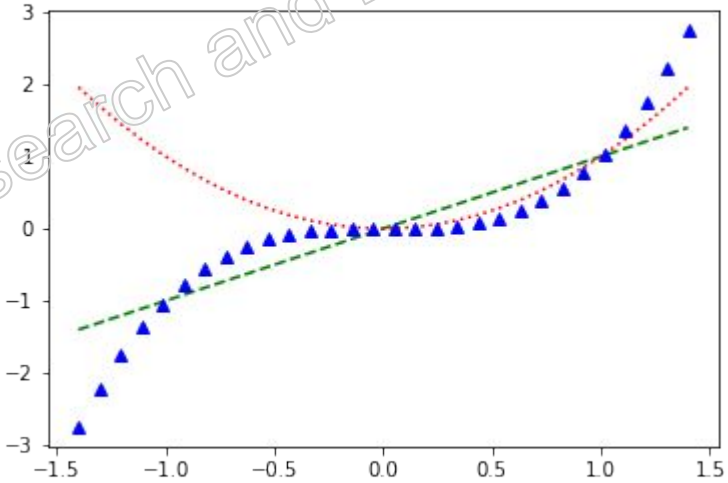
Matplotlib

```
x = np.linspace(-2, 2, 500)
y = x**2

plt.plot(x, y)
plt.title("Square function")
plt.xlabel("x")
plt.ylabel("y = x**2")
plt.grid(True)
plt.show()
```



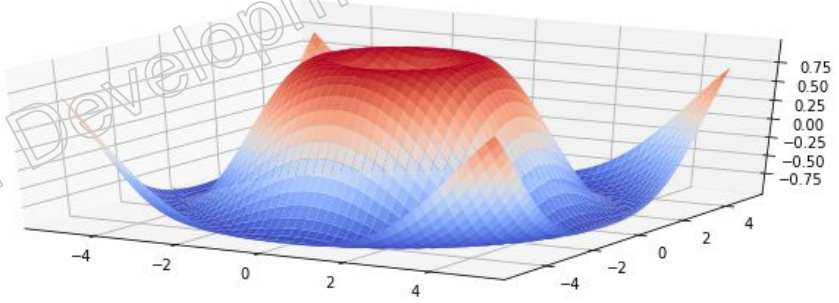
Matplotlib

```
x = np.linspace(-1.4, 1.4, 30)
plt.plot(x, x, 'g--', x, x**2, 'r:', x, x**3, 'b^')
plt.show()
```



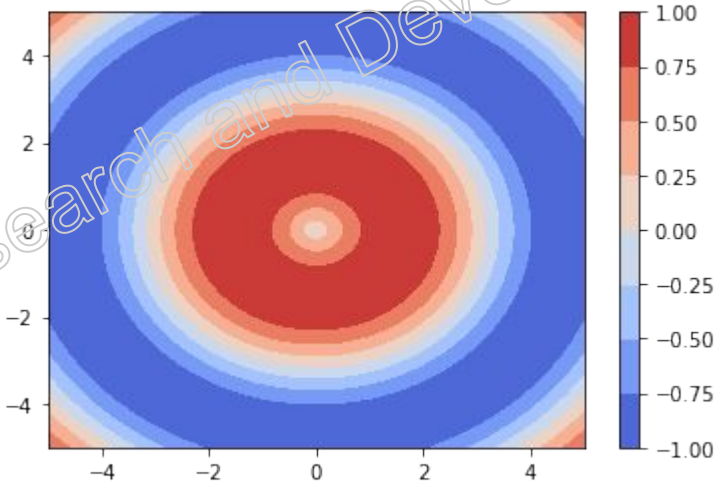
Matplotlib

```
from mpl_toolkits.mplot3d import Axes3D

x = np.linspace(-5, 5, 50)
y = np.linspace(-5, 5, 50)
X, Y = np.meshgrid(x, y)
R = np.sqrt(X**2 + Y**2)
Z = np.sin(R)

figure = plt.figure(1, figsize = (12, 4))
subplot3d = plt.subplot(111, projection='3d')
surface = subplot3d.plot_surface(X, Y, Z, rstride=1, cstride=1,
                                 cmap=matplotlib.cm.coolwarm, linewidth=0.1)
plt.show()
```



Matplotlib

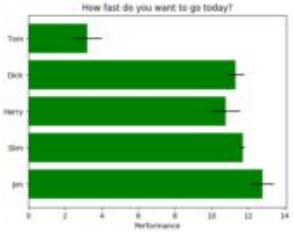
```
plt.contourf(X, Y, Z, cmap=matplotlib.cm.coolwarm)
plt.colorbar()
plt.show()
```



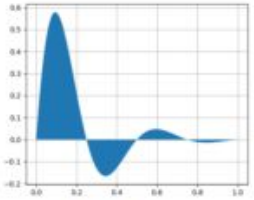
Matplotlib

Matplotlib - Gallery (https://matplotlib.org/gallery.html)

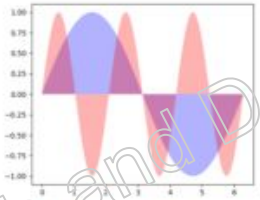
Lines, bars, and markers



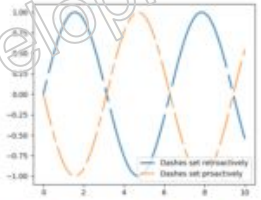
barh_demo



fill_demo

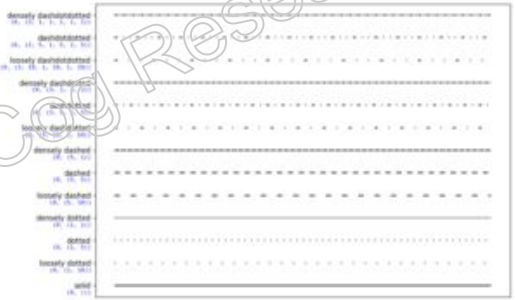


fill_demo_features

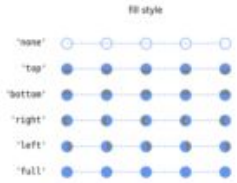


line_demo_dash_control

line_styles_reference



linestyles

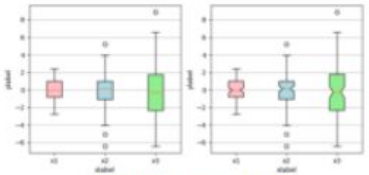


marker_fillstyle_reference

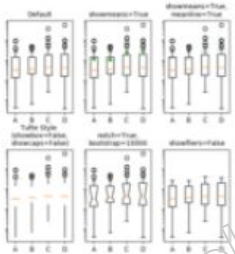
MATPLOTLIB

Matplotlib - Gallery (https://matplotlib.org/gallery.html)

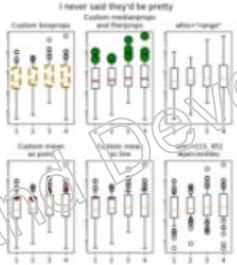
Statistical plots



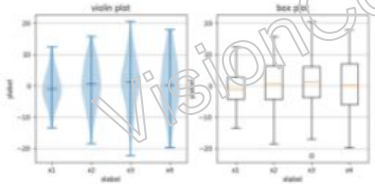
boxplot_color_demo



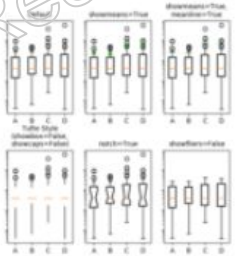
boxplot_demo



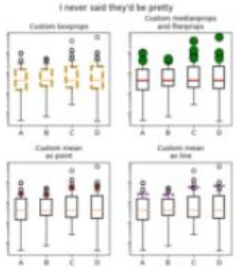
boxplot_demo



boxplot_vs_violin_demo



bxp_demo

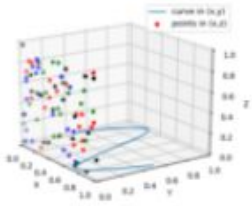


bxp_demo

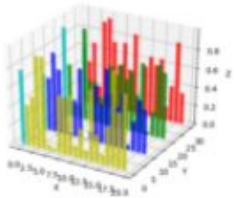
Matplotlib

Matplotlib - Gallery (https://matplotlib.org/gallery.html)

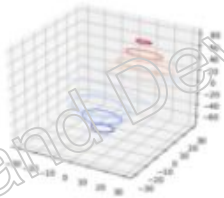
mplot3d toolkit



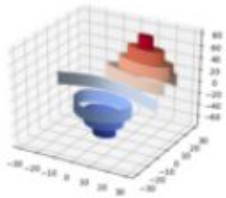
2dcollections3d_demo



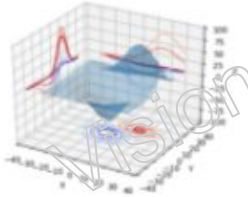
bars3d_demo



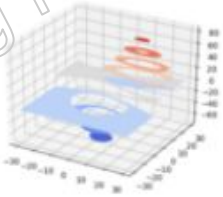
contour3d_demo



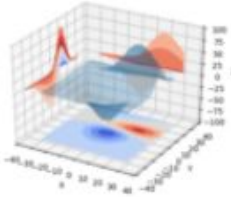
contour3d_demo2



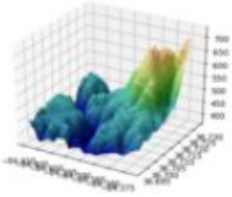
contour3d_demo3



contourf3d_demo



contourf3d_demo2



custom_shaded_3d_surface