

* Constants

1-

from scipy import constants

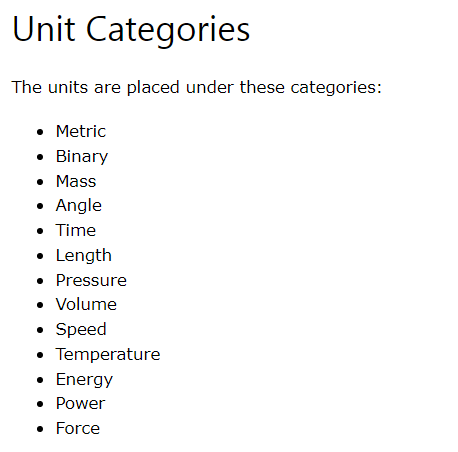
print(constants.pi)

2-

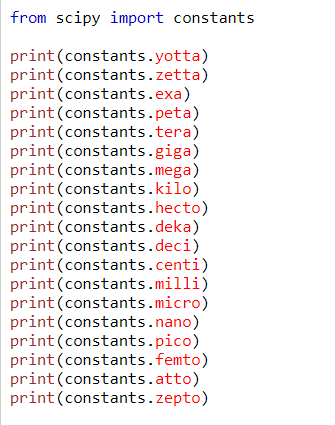
from scipy import constants

#Here, dir() is used to print list of contants.

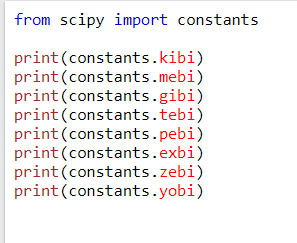
print(dir(constants))



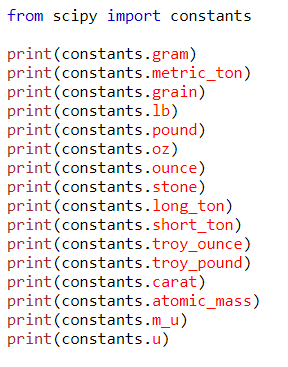
* Metric



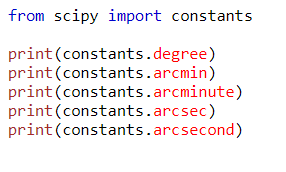
* Binary Prefix



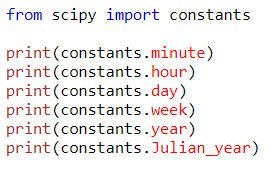
* Mass



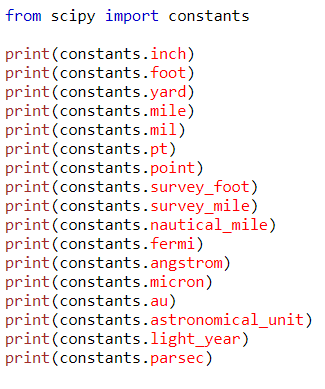
* Angle



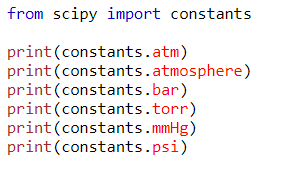
* Time



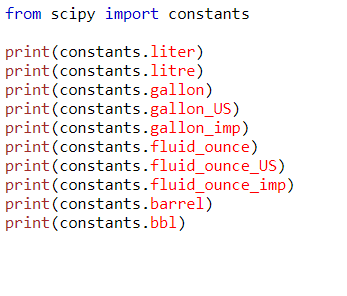
* Lenght



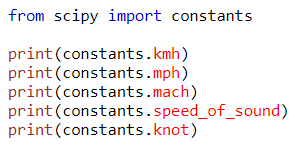
* Pressure



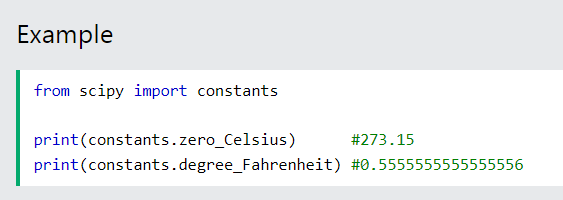
* Volume



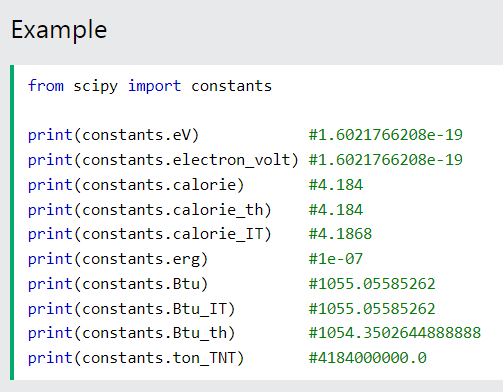
* Speed



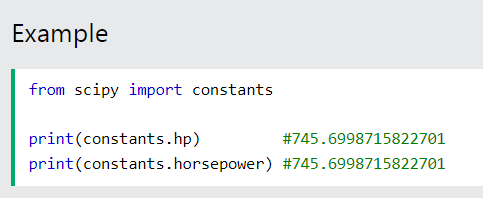
* Temp



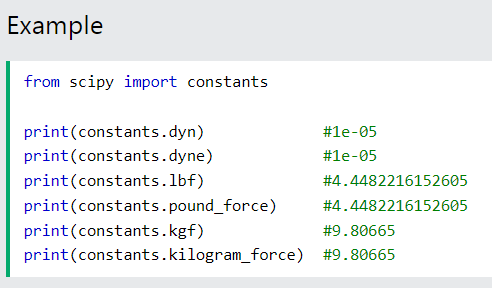
* Energy



* Hp



* Force



* Optimizers

1-

from scipy.optimize import root

from math import cos

def equ(x):

    return [x + cos(x)] #Here, we make it vector.

myroot = root(equ, 0)#Here, 0 is an initial guess.

#Here,if in future myroot.x[0] is an array and we are accessing it's first value at index 0.

print(f"Root found at x: {myroot.x[0]:.5f}")

2- Minimize a func

Note:

The minimize() function takes the following arguments:

fun - a function representing an equation.

x0 - an initial guess for the root.

method - name of the method to use. Legal values:  
    'CG'  
    'BFGS'  
    'Newton-CG'  
    'L-BFGS-B'  
    'TNC'  
    'COBYLA'  
    'SLSQP'

from scipy.optimize import minimize

def equ(x):

    return x \*\* 2 + x + 2

mini = minimize(equ, 0, method = 'TNC')

print(mini)

* Sparse Data

Note:

There are primarily two types of sparse matrices that we use:

**CSC** - Compressed Sparse Column. For efficient arithmetic, fast column slicing.

**CSR** - Compressed Sparse Row. For fast row slicing, faster matrix vector products

1-

import numpy as np

from scipy.sparse import csr\_matrix

arr = np.array([0, 2, 3, 5, 1, 0, 5, 4, 0, 2])

print(csr\_matrix(arr))

2- Viewing stored data (not the zero items)

import numpy as np

from scipy.sparse import csr\_matrix

arr = np.array([[0, 2, 3], [5, 1, 0], [5, 4, 0, 2]])

sparse\_matrix = csr\_matrix(arr)

print(sparse\_matrix.data)

Another method

import numpy as np

from scipy.sparse import csr\_matrix

arr = np.array([[0, 2, 3], [5, 1, 0], [5, 4, 0, 2]])

sparse\_matrix = csr\_matrix(arr)

sparse\_matrix.eliminate\_zeros()

print(sparse\_matrix)

3-

import numpy as np

from scipy.sparse import csr\_matrix

arr = np.array([[0, 2, 3], [5, 1, 0], [5, 4, 0, 2]])

sparse\_matrix = csr\_matrix(arr)

#Here, it eliminates duplicates values.

sparse\_matrix.sum\_duplicates()

print(sparse\_matrix)

4- convert from csr to csc

import numpy as np

from scipy.sparse import csr\_matrix

arr = np.array([[0, 2, 3], [5, 1, 0], [5, 4, 0, 2]])

sparse\_matrix = csr\_matrix(arr)

sparse\_matrix.tocsc()

print(sparse\_matrix)

* Graphs

1-

import numpy as np

#Here, we use connected\_components func from scipy.sparse.csgraph module.

from scipy.sparse.csgraph import connected\_components

from scipy.sparse import csr\_matrix

arr = np.array([[0, 1, 2],

          [2, 3, 4],

          [5, 6, 7]

          ])

newArr = csr\_matrix(arr)

print(connected\_components(newArr))

2- find the shortest path b/w one point to other.

import numpy as np

from scipy.sparse.csgraph import  dijkstra

from scipy.sparse import csr\_matrix

arr = np.array([[0, 1, 2],

          [2, 3, 4],

          [5, 6, 7]

          ])

newArr = csr\_matrix(arr)

#Here, predecessors means it will return previous nodes along the shortest path.

#Here, indices specifics the starting node which is 0

print(dijkstra(newArr, return\_predecessors=True, indices = 0))

3- find the shortest path b/w all pair of elements.

import numpy as np

from scipy.sparse.csgraph import floyd\_warshall

from scipy.sparse import csr\_matrix

arr = np.array([[0, 1, 2],

          [2, 3, 4],

          [5, 6, 7]

          ])

newArr = csr\_matrix(arr)

print(floyd\_warshall(newArr, return\_predecessors=True))

4- to find shortest path from element 1 to 2 with given graph with a negative weight

import numpy as np

from scipy.sparse.csgraph import bellman\_ford

from scipy.sparse import csr\_matrix

arr = np.array([[0, 1, 2],

          [2, 3, 4],

          [5, 6, 7]

          ])

newArr = csr\_matrix(arr)

print(bellman\_ford(newArr, return\_predecessors=True, indices = 0))

5- Depth first Order

Note: This function takes following arguments:

1. the graph.
2. the starting element to traverse graph from.

import numpy as np

from scipy.sparse.csgraph import depth\_first\_order

from scipy.sparse import csr\_matrix

arr = np.array([[0, 1, 2],

          [2, 3, 4],

          [5, 6, 7]

          ])

newArr = csr\_matrix(arr)

print(depth\_first\_order(newArr, 1))

6- Breadth First Order

Note: This function takes following arguments:

the graph.

the starting element to traverse graph from.

import numpy as np

from scipy.sparse.csgraph import breadth\_first\_order

from scipy.sparse import csr\_matrix

arr = np.array([[0, 1, 2],

          [2, 3, 4],

          [5, 6, 7]

          ])

newArr = csr\_matrix(arr)

print(breadth\_first\_order(newArr, 1))

## Spatial Data (data in geometric space)

## 1-

## Generating triangulation

import numpy as np

from scipy.spatial import Delaunay

import matplotlib as plt

samples = np.array([[2, 1],

               [5, 1],

               [9,6]])

#Here, simplies() return the indices of the vertices.

simplices = Delaunay(samples).simplices

#Here, first samples rep x-coordinates second rep y-coordinates and simplices rep indices of vertices.

plt.triplot(samples[:,0], samples[:, 1], simplices)

plt.show()

## 2-

## Convex Hull is the smallest polygon to cover all the given points

## 

import numpy as np

from scipy.spatial import ConvexHull

import matplotlib as plt

samples = np.array([[2, 1],

               [5, 1],

               [9,6]])

hull = ConvexHull(samples)

hull\_indices = hull.simplices

plt.scatter(samples[: 0], samples[: 1])

for points in hull\_indices:

#Here, 'k-' rep a black solid line.

    #Here, it plot a solid black line corresponding points in samples.

    plt.plot(samples[points, 0], samples[points, 1], "k-")

plt.show()

## 3-

## KD Trees

**Note: The KDTree() method returns a KDTree object.**

**The query() method returns the distance to the nearest neighbor *and* the location of the neighbors.**

from scipy import KDTree

samples = [(1, 2), (2, 3), (3, 4), (4, 5)]

tree\_of\_KD = KDTree(samples)

#Here, query() contains the nearest neibour and index of nearest point.

result = tree\_of\_KD.query(1,1)

print(result)

## 4-

## Distance Matrix

from scipy.spatial.distance import euclidean

point1 = (8,4)

point2 = (4,7)

result = euclidean(point1, point2)

print(result)

## 5-

## CityBlock Distance

from scipy.spatial.distance import cityblock

point1 = (8,4)

point2 = (4,7)

result = cityblock(point1, point2)

print(result)

## 6-

## Cosine Distance

from scipy.spatial.distance import cosine

point1 = (8,4)

point2 = (4,7)

result = cosine(point1, point2)

print(result)

## 7-

## Hamming Distance

from scipy.spatial.distance import hamming

point1 = (False, True, True)

point2 = (True, False, True)

result = hamming(point1, point2)

print(result)

## Matlab Arrays

## 1-

## Exportin Data in matlab format

**Note:** The method takes the following parameters:

1. **filename** - the file name for saving data.
2. **mdict** - a dictionary containing the data.
3. **do\_compression** - a boolean value that specifies whether to compress the result or not. Default False.

#Here, io provide function of reading and writing data.

from scipy import io

import numpy as np

arr = np.arange(10)

#Here, arr.mat save arr as a matlab file and the variable names will be variables and contains the values of arr.

io.savemat('arr.mat', {"variables": arr})

## 2-

## Import Data from Matlab format

from scipy import io

import numpy as np

arr = np.arange(10)

#Export

io.savemat('arr.mat', {"variables": arr})

#Import

myData = io.loadmat('arr.maat')

print(myData)

## 3-

from scipy import io

import numpy as np

arr = np.arange(10)

#Export

io.savemat('arr.mat', {"variables": arr})

#Here, squeez\_me is done to make the array 1D in the output otherwise it will be 2D.

myData = io.loadmat('arr.maat', squeez\_me = True)

print(myData["variables"])

## Interpolation