Constant Units

A list of all units under the constants module can be seen using the dir() function.

The units are placed under these categories:

Metric Binary Mass Angle Time Length Pressure Volume Speed Temperature Energy Power Force

'''

In [1]:
```
from scipy import constants

print(dir(constants))
```

```
['Avogadro', 'Boltzmann', 'Btu', 'Btu_IT', 'Btu_th', 'ConstantWarning', 'G', 'Julian
_year', 'N_A', 'Planck', 'R', 'Rydberg', 'Stefan_Boltzmann', 'Wien', '__all__', '__b
uiltins__', '__cached__', '__doc__', '__file__', '__loader__', '__name__', '__packag
e__', '__path__', '__spec__', '_codata', '_constants', '_obsolete_constants', 'acr
e', 'alpha', 'angstrom', 'arcmin', 'arcminute', 'arcsec', 'arcsecond', 'astronomical
_unit', 'atm', 'atmosphere', 'atomic_mass', 'atto', 'au', 'bar', 'barrel', 'bbl', 'b
lob', 'c', 'calorie', 'calorie_IT', 'calorie_th', 'carat', 'centi', 'codata', 'const
ants', 'convert_temperature', 'day', 'deci', 'degree', 'degree_Fahrenheit', 'deka',
'dyn', 'dyne', 'e', 'eV', 'electron_mass', 'electron_volt', 'elementary_charge', 'ep
silon_0', 'erg', 'exa', 'exbi', 'femto', 'fermi', 'find', 'fine_structure', 'fluid_o
unce', 'fluid_ounce_US', 'fluid_ounce_imp', 'foot', 'g', 'gallon', 'gallon_US', 'gal
lon_imp', 'gas_constant', 'gibi', 'giga', 'golden', 'golden_ratio', 'grain', 'gram',
'gravitational_constant', 'h', 'hbar', 'hectare', 'hecto', 'horsepower', 'hour', 'h
p', 'inch', 'k', 'kgf', 'kibi', 'kilo', 'kilogram_force', 'kmh', 'knot', 'lambda2n
u', 'lb', 'lbf', 'light_year', 'liter', 'litre', 'long_ton', 'm_e', 'm_n', 'm_p', 'm
_u', 'mach', 'mebi', 'mega', 'metric_ton', 'micro', 'micron', 'mil', 'mile', 'mill
i', 'minute', 'mmHg', 'mph', 'mu_0', 'nano', 'nautical_mile', 'neutron_mass', 'nu2la
mbda', 'ounce', 'oz', 'parsec', 'pebi', 'peta', 'physical_constants', 'pi', 'pico',
'point', 'pound', 'pound_force', 'precision', 'proton_mass', 'psi', 'pt', 'short_to
n', 'sigma', 'slinch', 'slug', 'speed_of_light', 'speed_of_sound', 'stone', 'survey_
foot', 'survey_mile', 'tebi', 'tera', 'test', 'ton_TNT', 'torr', 'troy_ounce', 'troy
_pound', 'u', 'unit', 'value', 'week', 'yard', 'year', 'yobi', 'yocto', 'yotta', 'ze
bi', 'zepto', 'zero_Celsius', 'zetta']
```

In [3]:
```
# Time: Return the specified unit in seconds (e.g. hour returns 3600.0)
from scipy import constants

print(constants.minute)      #60.0
print(constants.hour)        #3600.0
print(constants.day)         #86400.0
print(constants.week)        #604800.0
print(constants.year)        #31536000.0
print(constants.Julian_year) #31557600.0
```

```
60.0
3600.0
86400.0
604800.0
31536000.0
31557600.0
```

In [6]:

```python
#Metric (SI) Prefixes: Return the specified unit in meter (e.g. centi returns 0.01)

from scipy import constants

print(constants.yotta)    #1e+24
print(constants.zetta)    #1e+21
print(constants.exa)      #1e+18
print(constants.peta)     #1000000000000000.0
print(constants.tera)     #1000000000000.0
print(constants.giga)     #1000000000.0
print(constants.mega)     #1000000.0
print(constants.kilo)     #1000.0
print(constants.hecto)    #100.0
print(constants.deka)     #10.0
print(constants.deci)     #0.1
print(constants.centi)    #0.01
print(constants.milli)    #0.001
print(constants.micro)    #1e-06
print(constants.nano)     #1e-09
print(constants.pico)     #1e-12
print(constants.femto)    #1e-15
print(constants.atto)     #1e-18
print(constants.zepto)    #1e-21
```

```
1e+24
1e+21
1e+18
1000000000000000.0
1000000000000.0
1000000000.0
1000000.0
1000.0
100.0
10.0
0.1
0.01
0.001
1e-06
1e-09
1e-12
1e-15
1e-18
1e-21
```

In [7]:

```python
# Binary Prefixes: Return the specified unit in bytes (e.g. kibi returns 1024)
from scipy import constants

print(constants.kibi)    #1024
print(constants.mebi)    #1048576
```

```python
print(constants.gibi)    #1073741824
print(constants.tebi)    #1099511627776
print(constants.pebi)    #1125899906842624
print(constants.exbi)    #1152921504606846976
print(constants.zebi)    #1180591620717411303424
print(constants.yobi)    #1208925819614629174706176
```

```
1024
1048576
1073741824
1099511627776
1125899906842624
1152921504606846976
1180591620717411303424
1208925819614629174706176
```

In [8]:

```python
# Mass: Return the specified unit in kg (e.g. gram returns 0.001)
from scipy import constants

print(constants.gram)        #0.001
print(constants.metric_ton)  #1000.0
print(constants.grain)       #6.479891e-05
print(constants.lb)          #0.45359236999999997
print(constants.pound)       #0.45359236999999997
print(constants.oz)          #0.028349523124999998
print(constants.ounce)       #0.028349523124999998
print(constants.stone)       #6.3502931799999995
print(constants.long_ton)    #1016.0469088
print(constants.short_ton)   #907.1847399999999
print(constants.troy_ounce)  #0.031103476799999998
print(constants.troy_pound)  #0.37324172159999996
print(constants.carat)       #0.0002
print(constants.atomic_mass) #1.66053904e-27
print(constants.m_u)         #1.66053904e-27
print(constants.u)           #1.66053904e-27
```

```
0.001
1000.0
6.479891e-05
0.45359236999999997
0.45359236999999997
0.028349523124999998
0.028349523124999998
6.3502931799999995
1016.0469088
907.1847399999999
0.031103476799999998
0.37324172159999996
0.0002
1.6605390666e-27
1.6605390666e-27
1.6605390666e-27
```

In [9]:

```python
# Angle: Return the specified unit in radians (e.g. degree returns 0.01745329251994
from scipy import constants

print(constants.degree)     #0.017453292519943295
```

```
print(constants.arcmin)     #0.0002908882086657216
print(constants.arcminute)  #0.0002908882086657216
print(constants.arcsec)     #4.84813681109536e-06
print(constants.arcsecond)  #4.84813681109536e-06
```

```
0.017453292519943295
0.0002908882086657216
0.0002908882086657216
4.84813681109536e-06
4.84813681109536e-06
```

In [11]:

```
# Length: Return the specified unit in meters (e.g. nautical_mile returns 1852.0)
from scipy import constants

print(constants.inch)              #0.0254
print(constants.foot)              #0.30479999999999996
print(constants.yard)              #0.9143999999999999
print(constants.mile)              #1609.3439999999998
print(constants.mil)               #2.5399999999999997e-05
print(constants.pt)                #0.00035277777777777776
print(constants.point)             #0.00035277777777777776
print(constants.survey_foot)       #0.3048006096012192
print(constants.survey_mile)       #1609.3472186944373
print(constants.nautical_mile)     #1852.0
print(constants.fermi)             #1e-15
print(constants.angstrom)          #1e-10
print(constants.micron)            #1e-06
print(constants.au)                #149597870691.0
print(constants.astronomical_unit) #149597870691.0
print(constants.light_year)        #9460730472580800.0
print(constants.parsec)            #3.0856775813057292e+16
```

```
0.0254
0.30479999999999996
0.9143999999999999
1609.3439999999998
2.5399999999999997e-05
0.00035277777777777776
0.00035277777777777776
0.3048006096012192
1609.3472186944373
1852.0
1e-15
1e-10
1e-06
149597870700.0
149597870700.0
9460730472580800.0
3.085677581491367e+16
```

In [ ]:

```
# Pressure: Return the specified unit in pascals (e.g. psi returns 6894.75729316836
from scipy import constants

print(constants.atm)         #101325.0
print(constants.atmosphere)  #101325.0
print(constants.bar)         #100000.0
print(constants.torr)        #133.32236842105263
```

```
print(constants.mmHg)         #133.32236842105263
print(constants.psi)          #6894.757293168361
```

In [12]:
```
# Area: Return the specified unit in square meters(e.g. hectare returns 10000.0)
from scipy import constants

print(constants.hectare) #10000.0
print(constants.acre)    #4046.8564223999992
```

```
10000.0
4046.8564223999992
```

In [13]:
```
# Volume: Return the specified unit in cubic meters (e.g. liter returns 0.001)
from scipy import constants

print(constants.liter)             #0.001
print(constants.litre)             #0.001
print(constants.gallon)            #0.0037854117839999997
print(constants.gallon_US)         #0.0037854117839999997
print(constants.gallon_imp)        #0.00454609
print(constants.fluid_ounce)       #2.9573529562499998e-05
print(constants.fluid_ounce_US)    #2.9573529562499998e-05
print(constants.fluid_ounce_imp)  #2.84130625e-05
print(constants.barrel)            #0.15898729492799998
print(constants.bbl)               #0.15898729492799998
```

```
0.001
0.001
0.0037854117839999997
0.0037854117839999997
0.00454609
2.9573529562499998e-05
2.9573529562499998e-05
2.84130625e-05
0.15898729492799998
0.15898729492799998
```

In [14]:
```
# Speed: Return the specified unit in meters per second (e.g. speed_of_sound return
from scipy import constants

print(constants.kmh)            #0.2777777777777778
print(constants.mph)            #0.44703999999999994
print(constants.mach)           #340.5
print(constants.speed_of_sound) #340.5
print(constants.knot)           #0.5144444444444445
```

```
0.2777777777777778
0.44703999999999994
340.5
340.5
0.5144444444444445
```

In [15]:
```
#Temperature: Return the specified unit in Kelvin (e.g. zero_Celsius returns 273.15
from scipy import constants

print(constants.zero_Celsius)      #273.15
print(constants.degree_Fahrenheit) #0.5555555555555556
```

```
273.15
0.5555555555555556
```

In [16]:

```python
#Energy: Return the specified unit in joules (e.g. calorie returns 4.184)
from scipy import constants

print(constants.eV)            #1.6021766208e-19
print(constants.electron_volt) #1.6021766208e-19
print(constants.calorie)       #4.184
print(constants.calorie_th)    #4.184
print(constants.calorie_IT)    #4.1868
print(constants.erg)           #1e-07
print(constants.Btu)           #1055.05585262
print(constants.Btu_IT)        #1055.05585262
print(constants.Btu_th)        #1054.3502644888888
print(constants.ton_TNT)       #4184000000.0
```

```
1.602176634e-19
1.602176634e-19
4.184
4.184
4.1868
1e-07
1055.05585262
1055.05585262
1054.3502644888888
4184000000.0
```

In [17]:

```python
# Power: Return the specified unit in watts (e.g. horsepower returns 745.6998715822
from scipy import constants

print(constants.hp)         #745.6998715822701
print(constants.horsepower) #745.6998715822701
```

```
745.6998715822701
745.6998715822701
```

In [18]:

```python
# Force: Return the specified unit in newton (e.g. kilogram_force returns 9.80665)
from scipy import constants

print(constants.dyn)             #1e-05
print(constants.dyne)            #1e-05
print(constants.lbf)             #4.4482216152605
print(constants.pound_force)     #4.4482216152605
print(constants.kgf)             #9.80665
print(constants.kilogram_force)  #9.80665
```

```
1e-05
1e-05
4.4482216152605
4.4482216152605
9.80665
9.80665
```

In [21]:

```python
## Finding Minima
'''
We can use scipy.optimize.minimize() function to minimize the function.
```

```
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'
    'SSQP'

callback - function called after each iteration of optimization.

options - a dictionary defining extra params:

{
     "disp": boolean - print detailed description
     "gtol": number - the tolerance of the error
  }
'''
```

Out[21]:

```
'\nWe can use scipy.optimize.minimize() function to minimize the function.\n\nThe minimize() function takes the following arguments:\n\nfun - a function representing an equation.\n\nx0 - an initial guess for the root.\n\nmethod - name of the method to use. Legal values:\n    \'CG\'\n    \'BFGS\'\n    \'Newton-CG\'\n    \'L-BFGS-B\'\n    \'TNC\'\n    \'COBYLA\'\n    \'SLSQP\'\n\ncallback - function called after each iteration of optimization.\n\noptions - a dictionary defining extra params:\n\n{\n     "disp": boolean - print detailed description\n     "gtol": number - the tolerance of the error\n  }\n'
```

In [20]:

```
#Minimize the function x^2 + x + 2 with BFGS:

from scipy.optimize import minimize

def eqn(x):
  return x**2 + x + 2

mymin = minimize(eqn, 0, method='BFGS')

print(mymin)
```

```
      fun: 1.75
 hess_inv: array([[0.50000001]])
      jac: array([0.])
  message: 'Optimization terminated successfully.'
     nfev: 8
      nit: 2
     njev: 4
   status: 0
  success: True
        x: array([-0.50000001])
```

In [4]:
```
# Create a CSR matrix from an array
import numpy as np
from scipy.sparse import csr_matrix

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

print(csr_matrix(arr))

####### Output: From the result we can see that there are 3 items with value.

# The 1. item is in row 0 position 5 and has the value 1.

# The 2. item is in row 0 position 6 and has the value 1.

# The 3. item is in row 0 position 8 and has the value 2.
```

```
  (0, 5)	1
  (0, 6)	1
  (0, 8)	2
```

In [25]:
```
# Viewing stored data (not the zero items) with the data property of sparse matrix
csr_matrix(arr).data
```

Out[25]:
```
array([1, 1, 2], dtype=int32)
```

In [26]:
```
# Counting nonzeros with the count_nonzero() method of sparse matrix
csr_matrix(arr).count_nonzero()
```

Out[26]:
```
3
```

In [ ]:
```
################################   Scipy Graphs  ##################################
```

In [5]:
```
################################### Connected Components #########################
# https://docs.scipy.org/doc/scipy/reference/generated/scipy.sparse.csgraph.connect
import numpy as np
from scipy.sparse.csgraph import connected_components
from scipy.sparse import csr_matrix

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

graph = csr_matrix(arr)
print(f'Graph is \n {graph}')
```

```
Graph is 
   (0, 1)	1
  (0, 2)	2
  (1, 0)	1
  (2, 0)	2
```

In [19]:
```
number_of_connected_components, labels = connected_components(csgraph=graph, direct
```

In [20]:
```
number_of_connected_components
```
Out[20]:
```
1
```
In [13]:
```
labels
```
Out[13]:
```
array([0, 0, 0])
```
In [ ]:
```
################################# Dijkstra (https://docs.scipy.org/doc/scipy/referen
#Use the dijkstra method to find the shortest path in a graph from one element to a
# It takes following arguments:
# return_predecessors: boolean (True to return whole path of traversal otherwise Fa
# indices: index of the element to return all paths from that element only.
# limit: max weight of path.
```
In [6]:
```
# Example: Find the shortest path from element 1 to 2 using dijkstra
import numpy as np
from scipy.sparse.csgraph import dijkstra
from scipy.sparse import csr_matrix

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

newarr = csr_matrix(arr)

print(dijkstra(newarr, return_predecessors=True, indices=0))
```
```
(array([0., 1., 2.]), array([-9999,     0,     0]))
```
In [ ]:

In [ ]: