Q. Create a function to print all the factors of an entered number.

In [1]:
```
def Factors(num):
    print(f"Factors of {num} are: ")
    for i in range(1,num+1):
        if num%i==0:
            print(i,end=" ")
```

In [2]:
```
# Function call:
Factors(12)
```

```
Factors of 12 are:
1 2 3 4 6 12
```

Pre-defined Functions:-

In [3]:
```
# 1. print(): to display the output
print('Hello everyone')
```

```
Hello everyone
```

In [4]:
```
# 2. len(): returns total no.of elements in a seq.
len([4,76,8,23,56,4,234,456,98])
```

Out[4]: 9

In [5]:
```
# 3. sum(): returns the sum of all elements in a seq.
sum([5,32,56,8,234,56,78,9234,768])
```

Out[5]: 10471

In [6]:
```
# min(): returns the minimum value from seq.
min([5,32,56,8,234,56,78,9234,768])
```

Out[6]: 5

In [7]:
```
# max(): returns the maximum value from the seq.
max([5,32,56,8,234,56,78,9234,768])
```

Out[7]: 9234

In [8]:
```
# type(): returns the datatype of a variable.
a = 'apple'
type(a)
```

Out[8]: str

In [9]:
```
# id(): returns the address location of any variable.
a = [5,32,56,8,234,56,78,9234,768]
id(a)
```

Out[9]: 1870557120064

In [10]:
```
# range(start, stop+1, step):
list(range(1,11))
```

Out[10]: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

In [13]:
```
# ord(character):returns ascii value of the character.
ord('^')
```

Out[13]: 94

In [15]:
```
# chr(ascii_value): returns character associated with given ascii value.
chr(80)
```

Out[15]: 'P'

In [16]:
```
# pow(base,expo):
pow(25,2)
```

Out[16]: 625

NumPy:- Numerical Python

NumPy is a library of python that helps us to deal with arrays and to do various mathematical operations on it.

Array is a collection of homogenous value(having similar datas), mutable(can be modified/changed), which is declared using []. Arrays are an ordered sequence(have fixed index position).

In [17]:
```
# importing the library to our environment.
import numpy as np
```

In [18]:
```
# to check the version of numpy:-
np.__version__
```

Out[18]: '1.26.4'

In [19]:
```
# Creating a 1D array:
a = np.array([1,2,3,4,5])
a
```

Out[19]: array([1, 2, 3, 4, 5])

In [20]:
```
# type():
type(a)
```

Out[20]: numpy.ndarray

In [21]:
```
# Creating an array from tuple:
tup = (45,6,7,32,67,9)
type(tup)
```

Out[21]: tuple

In [22]:
```
b = np.array(tup)
b
```

Out[22]: array([45, 6, 7, 32, 67, 9])

In [25]:
```
c = np.array((1,2,6,'apple'))
c
```

Out[25]: array(['1', '2', '6', 'apple'], dtype='<U11')

In [26]:
```
# Creating array from dictionary:
d = {1:'a',2:'b',3:'c'}
x = np.array(d)
x
```

Out[26]: array({1: 'a', 2: 'b', 3: 'c'}, dtype=object)

In [27]:
```
# Creating an array from set:
np.array({'a','b','c'})
```

Out[27]: array({'b', 'a', 'c'}, dtype=object)

In [28]:
```
# Creating a 2D array:-
d2 = np.array([[1,2,3,4],[5,6,7,8],[9,10,11,12]])
d2
```

Out[28]:
```
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9, 10, 11, 12]])
```

In [29]:
```
# 3X3 matrix using tuple:-
np.array([(1,2,3),(5,76,78),(98,56,23)])
```

Out[29]:
```
array([[ 1,  2,  3],
       [ 5, 76, 78],
       [98, 56, 23]])
```

In [30]:
```
np.array([[2,3,4],(6,7,8)])
```

Out[30]:
```
array([[2, 3, 4],
       [6, 7, 8]])
```

In [31]:
```
# Array attributes:-
# to find the shape:
x = np.array([[2,3,4],(6,7,8)])
x.shape
```

Out[31]: (2, 3)

In [32]:
```
# to find the size of an array:
x = np.array([[2,3,4],(6,7,8)])
x.size
```

Out[32]: 6

In [33]:
```
# to find the dimension of the array:
x = np.array([[2,3,4],(6,7,8)])
x.ndim
```

Out[33]: 2

In [34]:
```
# creating an array of diff. lengths.
np.array([[1,2,3,4],[4,5,6,7],[6,7,8]])
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[34], line 2
      1 # creating an array of diff. lengths.
----> 2 np.array([[1,2,3,4],[4,5,6,7],[6,7,8]])

ValueError: setting an array element with a sequence. The requested array has an inh
omogeneous shape after 1 dimensions. The detected shape was (3,) + inhomogeneous par
t.
```

In [35]:
```
# arange(): generates numbers between the given interval.
np.arange(1,11)
```

Out[35]: array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])

In [36]:
```
# create an array of 3X2 where all values are 0.
np.array([[0,0],[0,0],[0,0]])
```

Out[36]:
```
array([[0, 0],
       [0, 0],
       [0, 0]])
```

In [38]:
```
# np.zeros: generates an 0's array.
np.zeros(5)
```

Out[38]: array([0., 0., 0., 0., 0.])

In [41]:
```
z = np.zeros((4,5))
z
```

Out[41]:
```
array([[0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.]])
```

In [42]:
```
z.dtype
```

Out[42]: dtype('float64')

In [43]:
```
np.zeros((5,5),dtype='int')
```

Out[43]:
```
array([[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]])
```

In [44]:
```
np.zeros((5,5),dtype=int)
```

Out[44]:
```
array([[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]])
```

In [53]:
```
x = np.zeros((5,5))
x.astype(int)
```

Out[53]:
```
array([[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]])
```

In [55]:
```
# ones:- creates an array with all values as 1.
np.ones(10)
```

Out[55]:
```
array([1., 1., 1., 1., 1., 1., 1., 1., 1., 1.])
```

In [56]:
```
np.ones((6,5),dtype=int)
```

Out[56]:
```
array([[1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1]])
```

In [59]:
```
# full():- creates an array where values are specific.
np.full((5,5),500)
```

Out[59]:
```
array([[500, 500, 500, 500, 500],
       [500, 500, 500, 500, 500],
       [500, 500, 500, 500, 500],
       [500, 500, 500, 500, 500],
       [500, 500, 500, 500, 500]])
```

In [60]:
```
np.full((5,5),'🍕')
```

Out[60]:
```
array([['🍕', '🍕', '🍕', '🍕', '🍕'],
       ['🍕', '🍕', '🍕', '🍕', '🍕'],
       ['🍕', '🍕', '🍕', '🍕', '🍕'],
       ['🍕', '🍕', '🍕', '🍕', '🍕'],
       ['🍕', '🍕', '🍕', '🍕', '🍕']], dtype='<U1')
```

In [61]:
```
# linspace(): generates equally distributed values.
np.linspace(1,100,10)
```

Out[61]:
```
array([  1.,  12.,  23.,  34.,  45.,  56.,  67.,  78.,  89., 100.])
```

In [63]:
```
# np.linspace(startrange, endrange, no.of values)
np.linspace(1,500,100)
```

Out[63]:
```
array([  1.        ,   6.04040404,  11.08080808,  16.12121212,
        21.16161616,  26.2020202 ,  31.24242424,  36.28282828,
        41.32323232,  46.36363636,  51.4040404 ,  56.44444444,
        61.48484848,  66.52525253,  71.56565657,  76.60606061,
        81.64646465,  86.68686869,  91.72727273,  96.76767677,
       101.80808081, 106.84848485, 111.88888889, 116.92929293,
       121.96969697, 127.01010101, 132.05050505, 137.09090909,
       142.13131313, 147.17171717, 152.21212121, 157.25252525,
       162.29292929, 167.33333333, 172.37373737, 177.41414141,
       182.45454545, 187.49494949, 192.53535354, 197.57575758,
       202.61616162, 207.65656566, 212.6969697 , 217.73737374,
       222.77777778, 227.81818182, 232.85858586, 237.8989899 ,
       242.93939394, 247.97979798, 253.02020202, 258.06060606,
       263.1010101 , 268.14141414, 273.18181818, 278.22222222,
       283.26262626, 288.3030303 , 293.34343434, 298.38383838,
       303.42424242, 308.46464646, 313.50505051, 318.54545455,
       323.58585859, 328.62626263, 333.66666667, 338.70707071,
       343.74747475, 348.78787879, 353.82828283, 358.86868687,
       363.90909091, 368.94949495, 373.98989899, 379.03030303,
       384.07070707, 389.11111111, 394.15151515, 399.19191919,
       404.23232323, 409.27272727, 414.31313131, 419.35353535,
       424.39393939, 429.43434343, 434.47474747, 439.51515152,
       444.55555556, 449.5959596 , 454.63636364, 459.67676768,
       464.71717172, 469.75757576, 474.7979798 , 479.83838384,
       484.87878788, 489.91919192, 494.95959596, 500.        ])
```

In [65]:
```
# identity matrix:-
np.eye(6,dtype=int)
```

Out[65]:
```
array([[1, 0, 0, 0, 0, 0],
       [0, 1, 0, 0, 0, 0],
       [0, 0, 1, 0, 0, 0],
       [0, 0, 0, 1, 0, 0],
       [0, 0, 0, 0, 1, 0],
       [0, 0, 0, 0, 0, 1]])
```

In [66]:
```
np.identity(7,dtype=int)
```

Out[66]:
```
array([[1, 0, 0, 0, 0, 0, 0],
       [0, 1, 0, 0, 0, 0, 0],
       [0, 0, 1, 0, 0, 0, 0],
       [0, 0, 0, 1, 0, 0, 0],
       [0, 0, 0, 0, 1, 0, 0],
       [0, 0, 0, 0, 0, 1, 0],
       [0, 0, 0, 0, 0, 0, 1]])
```

In [71]:
```
# Random Numbers in array.
# random.rand(): generates a random no between 0~1.
np.random.rand()
```

Out[71]: 0.11033519015534721

In [73]:
```
# creating a 1d array with random numbers.
np.random.rand(6)
```

Out[73]:
```
array([0.56333819, 0.98397395, 0.88572679, 0.39112588, 0.82182614,
       0.52586592])
```

In [74]:
```
np.random.rand(3,3)
```

Out[74]:
```
array([[0.0042271 , 0.62290984, 0.45233722],
       [0.47373505, 0.67721447, 0.24044775],
       [0.6507754 , 0.54312107, 0.62488854]])
```

In [76]:
```
# random.randint():generates random integer value.
np.random.randint(1,20)
```

Out[76]: 2

In [77]:
```
np.random.randint(1,20,(4,5))
```

Out[77]:
```
array([[ 8, 13, 15,  2, 13],
       [ 9, 10, 17, 10, 15],
       [ 9, 15, 18,  7, 17],
       [ 6,  9, 11, 19,  4]])
```

In [78]:
```
# Mathematical Operations on Arrays:-
a = np.array([3,45,6,8,9])
b = np.array([10,40,65,2,12])
print(a)
print(b)
```

```
[ 3 45  6  8  9]
[10 40 65  2 12]
```

In [79]:
```
# add():
np.add(a,b)
```

Out[79]: array([13, 85, 71, 10, 21])

In [80]:
```
# subtract():
np.subtract(a,b)
```

Out[80]: array([-7,   5, -59,   6, -3])

In [82]:
```
# cannot perform any operations on arrays having diff. length
a = np.array([3,45,6,8,9])
b = np.array([10,40,65,2,12,13])
np.add(a,b)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[82], line 4
      2 a = np.array([3,45,6,8,9])
      3 b = np.array([10,40,65,2,12,13])
----> 4 np.add(a,b)

ValueError: operands could not be broadcast together with shapes (5,) (6,)
```

In [83]:
```
# multiply:
a = np.array([3,45,6,8,9])
b = np.array([10,40,65,2,12])
np.multiply(a,b)
```

Out[83]:
```
array([  30, 1800,  390,   16,  108])
```

In [85]:
```
# divide(): returns quotient with decimal value
np.divide(a,b)
```

Out[85]:
```
array([0.3       , 1.125     , 0.09230769, 4.        , 0.75      ])
```

In [86]:
```
# floor_divide():
np.floor_divide(a,b)
```

Out[86]:
```
array([0, 1, 0, 4, 0])
```

In [87]:
```
# mod():
np.mod(a,b)
```

Out[87]:
```
array([3, 5, 6, 0, 9])
```

In [88]:
```
# divmod():
np.divmod(a,b)
```

Out[88]:
```
(array([0, 1, 0, 4, 0]), array([3, 5, 6, 0, 9]))
```

In [89]:
```
# sqrt():
np.sqrt(a)
```

Out[89]:
```
array([1.73205081, 6.70820393, 2.44948974, 2.82842712, 3.        ])
```

In [91]:
```
# power(array, expo):
np.power(b,2)
```

Out[91]:
```
array([ 100, 1600, 4225,    4,  144], dtype=int32)
```

In [92]:
```
# log():
np.log(a)
```

Out[92]:
```
array([1.09861229, 3.80666249, 1.79175947, 2.07944154, 2.19722458])
```

In [93]:
```
# sin():
np.sin(b)
```

Out[93]:
```
array([-0.54402111,  0.74511316,  0.82682868,  0.90929743, -0.53657292])
```

In [94]:
```
# exp():
np.exp(a)
```

Out[94]:
```
array([2.00855369e+01, 3.49342711e+19, 4.03428793e+02, 2.98095799e+03,
       8.10308393e+03])
```

In [95]:
```
# Statistical Operations on Arrays:
b = np.array([10,40,65,2,12])
b
```

Out[95]:
```
array([10, 40, 65,  2, 12])
```

In [96]:
```
# max():
np.max(b)
```

Out[96]:
```
65
```

In [97]:
```
# min():
np.min(b)
```

Out[97]:
```
2
```

In [98]:
```
# mean(): average
np.mean(b)
```

Out[98]:
```
25.8
```

In [99]:
```
# median():
np.median(b)
```

Out[99]:
```
12.0
```

In [100…
```
# argmax(): returns the index of maximum value
np.argmax(b)
```

Out[100…
```
2
```

In [101…
```
# argmin(): returns the index of minimum value.
np.argmin(b)
```

Out[101…
```
3
```

In [102…
```
# std(): Standard Deviation
np.std(b)
```

Out[102…
```
23.429895432971957
```

In [103…
```
# cumsum: Cummulative Summation
b = np.array([10,40,65,2,12])
np.cumsum(b)
```

Out[103… array([10, 50, 115, 117, 129])

In [104…
```
# cumprod: Cummulative Product
np.cumprod(b)
```

Out[104… array([10,   400, 26000, 52000, 624000])

In [105…
```
# cumproduct:
np.cumproduct(b)
```

Out[105… array([10,   400, 26000, 52000, 624000])

In [106…
```
# repeat():
b = np.array([10,40,65,2,12])
b
```

Out[106… array([10, 40, 65, 2, 12])

In [108…
```
# repeats each element for the given no.of times.
np.repeat(b,3)
```

Out[108… array([10, 10, 10, 40, 40, 40, 65, 65, 65, 2, 2, 2, 12, 12, 12])

In [110…
```
# tile():repeats the whole seq for given no.of times.
np.tile(b,3)
```

Out[110… array([10, 40, 65, 2, 12, 10, 40, 65, 2, 12, 10, 40, 65, 2, 12])

In [113…
```
# where():returns the index of specific element.
x = np.array([1,2,34,3,5,4,5,4,3,7,3,9,3])
np.where(x==3)
```

Out[113… (array([3, 8, 10, 12], dtype=int64),)

In [ ]: