Introduction to Functions & Numpy Library

by elham heidari

Basic Math Functions

In [1]:
```
#Import math library
import math
```

In [2]:
```
math.sqrt(25)
```

Out[2]:
```
5.0
```

In [3]:
```
math.log(100, 10)
```

Out[3]:
```
2.0
```

Basic Stat Functions

In [4]:
```
#Import statistics module
import statistics as stat
```

In [5]:
```
l = [12.1, 21.0, 13.4, 25.7, 30.1, 12.1, 12.1]
l
```

Out[5]:
```
[12.1, 21.0, 13.4, 25.7, 30.1, 12.1, 12.1]
```

In [6]:
```
stat.mean(l)
```

Out[6]:
```
18.071428571428573
```

In [7]:
```
stat.median(l)
```

Out[7]:
```
13.4
```

In [8]:
```
stat.mode(l)
```

Out[8]:
```
12.1
```

In [9]:
```
stat.stdev(l)
```

Out[9]:
```
7.530540991127239
```

In [10]:
```
stat.variance(l)
```

Out[10]:
```
56.709047619047624
```

Functions

Create function

Example: PV = FV/(1+r) ^ n

In [11]:
```
def pv1(fv, r, n):
    pv = fv / (1 + r) ** n
    return pv
```

In [12]:
```python
pv1(10000, 0.25, 5)
```
Out[12]:
```
3276.8
```

In [13]:
```python
type(pv1)
```
Out[13]:
```
function
```

In [14]:
```python
pv1(fv = 10000, r = 0.25, n = 5)
```
Out[14]:
```
3276.8
```

In [15]:
```python
pv1(n = 5, r = 0.25, fv = 10000)
```
Out[15]:
```
3276.8
```

In [16]:
```python
pv1(10000, 0.25) #error
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[16], line 1
----> 1 pv1(10000, 0.25)

TypeError: pv1() missing 1 required positional argument: 'n'
```

In [17]:
```python
#Arguments w/ default values
#Any number of arguments in a function can have a default value.
#But once we have a default argument, all the arguments to its right must also have default values.
def pv2(fv, r = 0.25, n):
    pv = fv / (1 + r) ** n
    return pv #error
```

```
  Cell In[17], line 4
    def pv2(fv, r = 0.25, n):
                          ^
SyntaxError: non-default argument follows default argument
```

In [18]:
```python
def pv2(fv, n, r = 0.25):
    pv = fv / (1 + r) ** n
    return pv

pv2(10000, 5)
```
Out[18]:
```
3276.8
```

In [19]:
```python
#local variables
n #error
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[19], line 2
      1 #local variables
----> 2 n

NameError: name 'n' is not defined
```

In [20]:
```python
%whos
```

```
Variable   Type        Data/Info
--------------------------------
l          list        n=7
math       module      <module 'math' (built-in)>
pv1        function    <function pv1 at 0x0000022C2081D120>
pv2        function    <function pv2 at 0x0000022C2232FD90>
stat       module      <module 'statistics' from<...>da3\\lib\\statistics.py'>
```

In [21]:
```
def pv3(fv, r = 0.25):
    n = 5
    pv = fv / (1 + r) ** n
    return pv

pv3(10000)
```

Out[21]: 3276.8

In [22]:
```
n = 5
def pv4(fv, r = 0.25):
    pv = fv / (1 + r) ** n
    return pv

pv4(10000)
```

Out[22]: 3276.8

Lambda Functions

In [23]:
```
(lambda x: x ** 2)(2)
```

Out[23]: 4

Example: Newton-Raphson Method

Solve f(x) = 0

f(x) = 4 * ln(x) - x

In [24]:
```
#Define function
import math

def f(x):
    return 4 * math.log(x) - x

def g(x):
    return 4 / x - 1
```

In [25]:
```
#Initial value
x0 = 0.5
```

In [26]:
```
all_x = [x0]
```

In [27]:
```
diff = 1
while diff > 0.0001:
    x1 = x0 - f(x0) / g(x0)
    all_x.append(x1)
    x0 = x1
    diff = abs(all_x[-1] - all_x[-2])
```

In [28]:
```
all_x
```

Out[28]:
```
[0.5,
 0.967512674605683,
 1.3183454639828363,
 1.4229788501445464,
 1.4295879035094685,
 1.429611824414113]
```

In [29]:
```
f(all_x[-1])
```

Out[29]: -5.599620767071656e-10

Numpy

NumPy (Numerical Python) is one of the core packages for numerical computing in Python.

Pandas, Matplotlib, Statmodels and many other Scientific libraries rely on NumPy.

Check NumPy documentation: [https://numpy.org/doc/stable/user/absolute_beginners.html (https://numpy.org/doc/stable/user/absolute_beginners.html)](https://numpy.org/doc/stable/user/absolute_beginners.html)

In [30]:
```
import numpy as np
```

In [31]:
```
#One-dimension array
a1 = np.array([3.4, 2.1, 5.8, 9.7, 4, 5.5])
a1
```

Out[31]:
```
array([3.4, 2.1, 5.8, 9.7, 4. , 5.5])
```

In [32]:
```
type(a1)
```

Out[32]:
```
numpy.ndarray
```

In [33]:
```
#Shape
a1.shape
```

Out[33]:
```
(6,)
```

In [34]:
```
#Indexing
a1[0]
```

Out[34]:
```
3.4
```

In [35]:
```
a1[1 : 4]
```

Out[35]:
```
array([2.1, 5.8, 9.7])
```

In [36]:
```
a1[[0, 3, 4]]
```

Out[36]:
```
array([3.4, 9.7, 4. ])
```

In [37]:
```
a1[-1]
```

Out[37]:
```
5.5
```

In [38]:
```
#Matrix
a2 = np.array([[1, 3, 5], [0, 2, 6], [3, 7, 9]])
a2
```

Out[38]:
```
array([[1, 3, 5],
       [0, 2, 6],
       [3, 7, 9]])
```

In [39]:
```
#Shape
a2.shape
```

Out[39]:
```
(3, 3)
```

In [40]:
```
#Indexing
a2[0, 1]
```

Out[40]:
```
3
```

In [41]:
```
a2[0]
```

Out[41]:
```
array([1, 3, 5])
```

In [42]:
```
a2[0, :]
```
Out[42]:
```
array([1, 3, 5])
```
In [43]:
```
a2[:, 0]
```
Out[43]:
```
array([1, 0, 3])
```
In [44]:
```
a2[1 :, 1:]
```
Out[44]:
```
array([[2, 6],
       [7, 9]])
```
In [45]:
```
#Data type
a2.dtype
```
Out[45]:
```
dtype('int32')
```
In [46]:
```
#Modify an array
a1
```
Out[46]:
```
array([3.4, 2.1, 5.8, 9.7, 4. , 5.5])
```
In [47]:
```
a1[1] = 0
a1
```
Out[47]:
```
array([3.4, 0. , 5.8, 9.7, 4. , 5.5])
```
In [48]:
```
a2
```
Out[48]:
```
array([[1, 3, 5],
       [0, 2, 6],
       [3, 7, 9]])
```
In [49]:
```
a2[1, 2] = 0
a2
```
Out[49]:
```
array([[1, 3, 5],
       [0, 2, 0],
       [3, 7, 9]])
```
In [50]:
```
a3 = np.zeros((2, 3), dtype = int)
a3
```
Out[50]:
```
array([[0, 0, 0],
       [0, 0, 0]])
```
In [51]:
```
a4 = np.ones((3, 3), dtype = int)
a4
```
Out[51]:
```
array([[1, 1, 1],
       [1, 1, 1],
       [1, 1, 1]])
```
In [52]:
```
#Create ranges with arange
a5 = np.arange(1, 20, 3)
a5
```
Out[52]:
```
array([ 1,  4,  7, 10, 13, 16, 19])
```
In [53]:
```
a6 = np.arange(5)
a6
```
Out[53]:
```
array([0, 1, 2, 3, 4])
```

In [54]:
```python
#Create floating-point ranges with linspace
a7 = np.linspace(1, 5, 30)
a7
```

Out[54]:
```
array([1.        , 1.13793103, 1.27586207, 1.4137931 , 1.55172414,
       1.68965517, 1.82758621, 1.96551724, 2.10344828, 2.24137931,
       2.37931034, 2.51724138, 2.65517241, 2.79310345, 2.93103448,
       3.06896552, 3.20689655, 3.34482759, 3.48275862, 3.62068966,
       3.75862069, 3.89655172, 4.03448276, 4.17241379, 4.31034483,
       4.44827586, 4.5862069 , 4.72413793, 4.86206897, 5.        ])
```

In [55]:
```python
a7.shape
```

Out[55]:
```
(30,)
```

In [56]:
```python
a7 = a7.reshape(5, 6)
a7
```

Out[56]:
```
array([[1.        , 1.13793103, 1.27586207, 1.4137931 , 1.55172414,
        1.68965517],
       [1.82758621, 1.96551724, 2.10344828, 2.24137931, 2.37931034,
        2.51724138],
       [2.65517241, 2.79310345, 2.93103448, 3.06896552, 3.20689655,
        3.34482759],
       [3.48275862, 3.62068966, 3.75862069, 3.89655172, 4.03448276,
        4.17241379],
       [4.31034483, 4.44827586, 4.5862069 , 4.72413793, 4.86206897,
        5.        ]])
```

In [57]:
```python
a7.shape
```

Out[57]:
```
(5, 6)
```

In [58]:
```python
#Create a 3x3 array of normally distributed random values
# with mean 0 and standard deviation 1
np.random.seed(123)
a8 = np.random.normal(0, 1, (3, 3))
a8
```

Out[58]:
```
array([[-1.0856306 ,  0.99734545,  0.2829785 ],
       [-1.50629471, -0.57860025,  1.65143654],
       [-2.42667924, -0.42891263,  1.26593626]])
```

In [59]:
```python
#Create a 3x3 array of random integers in the interval [0, 10)
np.random.seed(123)
a9 = np.random.randint(0, 10, (3, 3))
a9
```

Out[59]:
```
array([[2, 2, 6],
       [1, 3, 9],
       [6, 1, 0]])
```

In [60]:
```python
#Summary statistics
a9
```

Out[60]:
```
array([[2, 2, 6],
       [1, 3, 9],
       [6, 1, 0]])
```

In [61]:
```python
np.mean(a9)
```

Out[61]:
```
3.3333333333333335
```

In [62]:
```python
np.std(a9)
```

Out[62]:
```
2.8284271247461903
```

In [63]:
```python
np.min(a9)
```

Out[63]:
```
0
```

In [64]:
```
np.max(a9)
```

Out[64]:
```
9
```

In [65]:
```
#Sum along rows (sum over columns)
np.sum(a9, axis = 0)
```

Out[65]:
```
array([ 9,  6, 15])
```

In [66]:
```
#Sum along columns (sum over rows)
np.sum(a9, axis = 1)
```

Out[66]:
```
array([10, 13,  7])
```

In [67]:
```
#Array operations
a9 + 5
```

Out[67]:
```
array([[ 7,  7, 11],
       [ 6,  8, 14],
       [11,  6,  5]])
```

In [68]:
```
a9 * 2
```

Out[68]:
```
array([[ 4,  4, 12],
       [ 2,  6, 18],
       [12,  2,  0]])
```

In [69]:
```
a2 + a9
```

Out[69]:
```
array([[ 3,  5, 11],
       [ 1,  5,  9],
       [ 9,  8,  9]])
```

In [70]:
```
a2 * a9
```

Out[70]:
```
array([[ 2,  6, 30],
       [ 0,  6,  0],
       [18,  7,  0]])
```

In [71]:
```
#Matrix multiplication
np.dot(a2, a9)
```

Out[71]:
```
array([[35, 16, 33],
       [ 2,  6, 18],
       [67, 36, 81]])
```

In [72]:
```
#Transpose of a matrix
a9.T
```

Out[72]:
```
array([[2, 1, 6],
       [2, 3, 1],
       [6, 9, 0]])
```

In [73]:
```
#Determinant of a matrix
np.linalg.det(a9)
```

Out[73]:
```
-11.999999999999995
```

In [74]:
```
#Boolian operations
a9 > 2
```

Out[74]:
```
array([[False, False,  True],
       [False,  True,  True],
       [ True, False, False]])
```

In [75]:
```
a9[a9 > 2]
```

Out[75]:
```
array([6, 3, 9, 6])
```

In [76]:
```
#  &: and
#  |: or
#  ~: not

a9[(a9 % 3 == 0) & (a9 > 4)]
```
Out[76]:
```
array([6, 9, 6])
```

In [77]:
```
a9[(a9 < 2) | (a9 > 6)]
```
Out[77]:
```
array([1, 9, 1, 0])
```

In [78]:
```
#axis = 0 : concatenate vertically
#axis = 1 : concatenate horizontally
np.concatenate((a2, a9), axis = 0)
```
Out[78]:
```
array([[1, 3, 5],
       [0, 2, 0],
       [3, 7, 9],
       [2, 2, 6],
       [1, 3, 9],
       [6, 1, 0]])
```

In [79]:
```
#Example: Consider a1,
# sort valuses of a1.
# find the index of maximum value
# find the index of minimum value
# find the rank of each item
# find the index of a value which equals to 5.8.
```

In [80]:
```
a1
```
Out[80]:
```
array([3.4, 0. , 5.8, 9.7, 4. , 5.5])
```

In [81]:
```
np.sort(a1)
```
Out[81]:
```
array([0. , 3.4, 4. , 5.5, 5.8, 9.7])
```

In [82]:
```
np.argmax(a1)
```
Out[82]:
```
3
```

In [83]:
```
np.argmin(a1)
```
Out[83]:
```
1
```

In [84]:
```
np.argsort(a1)
```
Out[84]:
```
array([1, 0, 4, 5, 2, 3], dtype=int64)
```

In [85]:
```
np.where(a1 == 5.8)
```
Out[85]:
```
(array([2], dtype=int64),)
```

Assignment

Q1: Write a function to find the sum of digits of an integer number.

Q2: Create a function that can solve any quadratic equation using Newton-Raphson method.

Q3: Create a 4×5 numpy array of all True’s.

Q4: Create an array of integers from 10 to 50 with with step of 3. Save the array in a. Replace all even numbers in a with -1.

Q5: Create an one-dimensional array which includes odd numbers from 1 to 20.

a: Change all numbers divisible by 3 into 0.

b: Change the array into a 2x5 matrix.

c: Convert all numbers of the third column into zero.

Q6:

a: Create an one-dimensional array, b1, which includes 9 numbers evenly spaced over 1.5 to 3.

b: Create a 3x3 matrix, b2, which includes random numbers from Gaussian distribution with mean = 1, and sd = 0.75, seed = 1234.

c: Calculate the determinant of b2.

d: What is the transpose of matrix b2?

e: Convert b1 into a 3x3 matrix.

f: Calculate summation of b1 and b2.

g: Calculate matrix multiplication of b1 and b2.

h: Call all numbers of b2 which are greater than mean of b2.

i: Scale matrix b2 using min-max scaling method.

End of Code