NumPy Workbook

May 10, 2022

1 NumPy Workbook

Welcome to the NumPy Workbook. Let's start with importing the NumPy library.

[1]:
```
import numpy as np
```

1.1 1.NumPy Arrays

Let's start with defining a NumPy array.

[2]:
```
array_a = np.array([1,2,3,4])
```

[3]:
```
print(array_a)
```

```
[1 2 3 4]
```

It is important to note that we have used square brackets inside the paranthesis to define the elements of the array.

[4]:
```
array_b = np.array([[1,2,3,4], [5,6,7,8]])
```

[5]:
```
print(array_b)
```

```
[[1 2 3 4]
 [5 6 7 8]]
```

Let's explore a few of the functionality with these arrays.

Using print and type functionalities.

[6]:
```
print('Type of array_a is: ', type(array_a))
```

```
Type of array_a is:  <class 'numpy.ndarray'>
```

[7]:
```
print('Type of array_b is: ', type(array_b))
```

```
Type of array_b is:  <class 'numpy.ndarray'>
```

Using the size, shape, ndim, len functionalities.

[8]:
```
print('Size of array_a is: ',array_a.size)
```

```
Size of array_a is:  4
```

[9]:
```
print('Size of array_b is: ', array_b.size)
```

```
Size of array_b is:  8
```

[10]:
```
print('Shape of array_a is: ', array_a.shape)
```

```
Shape of array_a is:  (4,)
```

[11]:
```
print('Shape of array_b is: ', array_b.shape)
```

```
Shape of array_b is:  (2, 4)
```

[12]:
```
print('Dimensions of array_a are: ', array_a.ndim)
```

```
Dimensions of array_a are:  1
```

[13]:
```
print('Dimensions of array_b are: ', array_b.ndim)
```

```
Dimensions of array_b are:  2
```

[14]:
```
print('Length of array_a is: ', len(array_a))
```

```
Length of array_a is:  4
```

[15]:
```
print('Length of array_b is: ', len(array_b))
```

```
Length of array_b is:  2
```

'len' function returned only the first dimension!

Understanding the index. Python uses 0 based indexing. Let's go through the exercises below to understand items and indices.

[16]:
```
print('Array_a is: \n', array_a)
```

```
Array_a is:
 [1 2 3 4]
```

[17]:
```
print('Array_b is: \n', array_b)
```

```
Array_b is:
 [[1 2 3 4]
 [5 6 7 8]]
```

[18]:
```
print('Index 1 of array_a is: ', array_a[1])
```

```
Index 1 of array_a is:  2
```

[19]:
```
print('Index 0 of array_a is: ', array_a[0])
```

```
Index 0 of array_a is:  1
```

[20]:
```
print('Index 4 of array_a is: ', array_a[4])
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
Input In [20], in <cell line: 1>()
----> 1 print('Index 4 of array_a is: ', array_a[4])

IndexError: index 4 is out of bounds for axis 0 with size 4
```

[21]:
```
print('Index 3 of array_a is: ', array_a[3])
```

```
Index 3 of array_a is:  4
```

[22]:
```
print('Last item of array_a is: ', array_a[-1])
```

```
Last item of array_a is:  4
```

[23]:
```
print('Index 4 of array_b is: ', array_b[4])
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
Input In [23], in <cell line: 1>()
----> 1 print('Index 4 of array_b is: ', array_b[4])

IndexError: index 4 is out of bounds for axis 0 with size 2
```

[24]:
```
print('4th item of array_b is: ', array_b[0,3])
```

```
4th item of array_b is:  4
```

[25]:
```
print('4th item of array_b is: ', array_b[0,-1])
```

```
4th item of array_b is:  4
```

[26]:
```
print('6th item of array_b is: ', array_b[1,1])
```

```
6th item of array_b is:  6
```

1.1.1 Creating NumPy arrays.

[28]:
```
# Create an array of shape 5,1 with zeros.
array_zeros = np.zeros(5)
print('array_zeros is: \n', array_zeros)
print('Shape of array_zeros is: ', array_zeros.shape)
print('Size of array_zeros is: ', array_zeros.size)
print('Dimensions of array_zeros is: ', array_zeros.ndim)
print('Length of array_zeros is: ', len(array_zeros))
```

```
array_zeros is:
 [0. 0. 0. 0. 0.]
Shape of array_zeros is:  (5,)
Size of array_zeros is:  5
Dimensions of array_zeros is:  1
Length of array_zeros is:  5
```

[30]:
```
# Create an array of shape 5,2 with ones.
array_ones = np.ones([5,2])
print('array_ones is: \n', array_ones)
print('Shape of array_ones is: ', array_ones.shape)
print('Size of array_ones is: ', array_ones.size)
print('Dimensions of array_ones is: ', array_ones.ndim)
print('Length of array_ones is: ', len(array_ones))
```

```
array_ones is:
 [[1. 1.]
 [1. 1.]
 [1. 1.]
 [1. 1.]
 [1. 1.]]
Shape of array_ones is:  (5, 2)
Size of array_ones is:  10
Dimensions of array_ones is:  2
Length of array_ones is:  5
```

[31]:
```
# Create an empty array of shape 5,2,3.
array_empty = np.empty([5,2,3])
print('array_empty is: \n', array_empty)
print('Shape of array_empty is: ', array_empty.shape)
print('Size of array_empty is: ', array_empty.size)
print('Dimensions of array_empty is: ', array_empty.ndim)
```

```
array_empty is:
 [[[2.00000000e+000 1.29073959e-231 9.38724727e-323]
  [0.00000000e+000 0.00000000e+000 5.02034658e+175]]

 [[1.36949091e-071 1.13070363e-042 1.12159792e-047]
```

```
  [7.24780137e-042 8.23824047e-043 5.19439565e-144]]

 [[3.59751658e+252 1.46901661e+179 8.37404147e+242]
  [2.59027926e-144 3.80985069e+180 5.70895899e+281]]

 [[2.59027912e-144 7.79952704e-143 9.91604872e+164]
  [2.00562045e-076 4.27345210e-033 1.32390754e-071]]

 [[2.59027860e-144 2.59903818e-144 5.88728852e-091]
  [6.76092015e-067 2.31738028e-052 1.11475752e+261]]]
Shape of array_empty is:  (5, 2, 3)
Size of array_empty is:  30
Dimensions of array_empty is:  3
```

We can also create an array in a range of evenly spaced intervals with the arange function. aranged_array = np.arange(first number, last number, step size). Let's go:

[32]:
```
# Create an array from 0 to 22, with increments of 3.
aranged_array = np.arange(0, 22, 3)
print('The aranged_array is: ', aranged_array)
print('The shape of the aranged_array is: ', aranged_array.shape)
```

```
The aranged_array is:  [ 0  3  6  9 12 15 18 21]
The shape of the aranged_array is:  (8,)
```

Similarly, we can create linearly spaced arrays using linspace function. spaced_array = np.linspace(first number, last number, number of intervals).

[33]:
```
# Create an array from 0 to 21, with 7 intervals.
spaced_array = np.linspace(0, 21, 8)
print('The spaced_array is: \n', spaced_array)
print('The shape of the spaced_array is: ',spaced_array.shape)
```

```
The spaced_array is:
 [ 0.  3.  6.  9. 12. 15. 18. 21.]
The shape of the spaced_array is:  (8,)
```

We can also reshape the arrays using .transpose() and .reshape() function.

[34]:
```
spaced_array_transpose = spaced_array.transpose()
print('The spaced_array is: \n', spaced_array)
print('The shape of the spaced_array is: ',spaced_array.shape)
print('The spaced_array_transpose is: \n', spaced_array_transpose)
print('The shape of the spaced_array_transpose is: ',spaced_array_transpose.
 ↪shape)
```

```
The spaced_array is:
 [ 0.  3.  6.  9. 12. 15. 18. 21.]
The shape of the spaced_array is:  (8,)
The spaced_array_transpose is:
```

```
 [ 0.  3.  6.  9. 12. 15. 18. 21.]
The shape of the spaced_array_transpose is:  (8,)
```

[35]:
```
spaced_array_reversed = np.reshape(spaced_array, newshape=(1,␣
 ↪len(spaced_array)))
print('The spaced_array_reversed is: \n', spaced_array_reversed)
print('The shape of the spaced_array_reversed is: ',spaced_array_reversed.shape)
```

```
The spaced_array_reversed is:
 [[ 0.  3.  6.  9. 12. 15. 18. 21.]]
The shape of the spaced_array_reversed is:  (1, 8)
```

[36]:
```
spaced_array_reshaped = np.reshape(spaced_array, newshape=(2, 4))
print('The spaced_array_reshaped is: \n', spaced_array_reshaped)
print('The shape of the spaced_array_reshaped is: ',spaced_array_reshaped.shape)
```

```
The spaced_array_reshaped is:
 [[ 0.  3.  6.  9.]
 [12. 15. 18. 21.]]
The shape of the spaced_array_reshaped is:  (2, 4)
```

[37]:
```
spaced_array_reshaped_transpose = np.reshape(spaced_array, newshape=(2, 4)).
 ↪transpose()
print('The spaced_array_reshaped_transpose is: \n',␣
 ↪spaced_array_reshaped_transpose)
print('The shape of the spaced_array_reshaped_transpose is:␣
 ↪',spaced_array_reshaped_transpose.shape)
```

```
The spaced_array_reshaped_transpose is:
 [[ 0. 12.]
 [ 3. 15.]
 [ 6. 18.]
 [ 9. 21.]]
The shape of the spaced_array_reshaped_transpose is:  (4, 2)
```

1.2 2.Basic NumPy Functions

[38]:
```
# Min, Max, Mean, Sum can be calculated easily.
sample_array = np.linspace(0,100,21).reshape(3,7)
print('Sample array is: \n', sample_array)
print('Min value of sample array is: ', sample_array.min())
print('Max value of sample array is: ', sample_array.max())
print('Mean value of sample array is: ', sample_array.mean())
print('Sum of sample array values is: ', sample_array.sum())
```

```
Sample array is:
 [[  0.   5.  10.  15.  20.  25.  30.]
 [ 35.  40.  45.  50.  55.  60.  65.]
```

```
 [ 70.  75.  80.  85.  90.  95. 100.]]
Min value of sample array is:  0.0
Max value of sample array is:  100.0
Mean value of sample array is:  50.0
Sum of sample array values is:  1050.0
```

[39]:
```
# Certain elements of an array can be accessd with "slicing".
print('Second row of sample array is: \n', sample_array[1,:])
```

```
Second row of sample array is:
 [35. 40. 45. 50. 55. 60. 65.]
```

1.3 3.Custom Functions with NumPy

[40]:
```
### Custom Functions can be built with the following structure:

def my_function(input):
    output = input*2 + 1
    return output
```

[41]:
```
a = 5
b = my_function(a)

print(b)
```

```
11
```

For Loops The elements of an array can be called separately through an iterative loop.

[42]:
```
sample_array = np.arange(0,20,5)
print('Sample array is : \n', sample_array)
```

```
Sample array is :
 [ 0  5 10 15]
```

How will we access each element on this array? There are two ways, we can directly call elements in an array or call the index of elements. Check out these two codes and see what is different.

[43]:
```
for item in (sample_array):
    print(item)
```

```
0
5
10
15
```

[44]:
```
for character in 'learning':
    print(character)
```

l
e
a
r
n
i
n
g

[46]:
```
for i in range(len(sample_array)):
    print(i,'th element of the sample array is: ', sample_array[i])
```

```
0 th element of the sample array is:  0
1 th element of the sample array is:  5
2 th element of the sample array is:  10
3 th element of the sample array is:  15
```

We can also perform computations on the sample array and store the new values to a new NumPy array.

[47]:
```
# First, let's create a new array of zeros, with the same length of␣
 ↪sample_array.
new_array = np.zeros(len(sample_array))
```

[48]:
```
for i in range(len(sample_array)):
    new_array[i] = my_function(sample_array[i])
```

[49]:
```
print('The values of new array are: \n', new_array)
```

```
The values of new array are:
 [ 1. 11. 21. 31.]
```

if/else/elif statements Another important computational tool is decision making. We can use if, else, and elif statement in Python to make logic based selections.

[51]:
```
a = 100
b = 50
c = 100

print(' a>=b: ', a>=b, '\n', 'b<a: ', b<=a, '\n', 'a==b: ', a==b, '\n', 'a==c:␣
 ↪', a==c, '\n', 'a!=b: ', a!=b, '\n', 'a!=c: ', a!=c)
```

```
 a>=b:  True
 b<a:  True
 a==b:  False
 a==c:  True
 a!=b:  True
 a!=c:  False
```

For example, let's start with a discrete function, where:

$$y(x) = 10x - 1, x > 0$$

$$y(x) = 1, x = 0$$

$$y(x) = 2x/5, x < 0$$

We can write this in Python as follows:

[52]:
```
def function1_for_y(x):
    if (x > 0):
        y = 10.*x - 1
    if (x == 0):
        y = 1.
    if (x < 0):
        y = 2.*x/5.

    return y
```

[54]:
```
def function2_for_y(x):
    if (x > 0):
        y = 10.*x - 1
    elif (x == 0):
        y = 1.
    else:
        y = 2.*x/5.
    return y
```

[55]:
```
print('Value -1: ', function1_for_y(-5), 'Value -2: ',function2_for_y(-5))
print('Value -1: ', function1_for_y(0), 'Value -2: ',function2_for_y(0))
print('Value -1: ', function1_for_y(5), 'Value -2: ',function2_for_y(5))
```

```
Value -1:  -2.0 Value -2:  -2.0
Value -1:  1.0 Value -2:  1.0
Value -1:  49.0 Value -2:  49.0
```

What happens when we try to pass an array?

[56]:
```
x = np.linspace(-10,10,11)
print(x)
```

```
[-10.  -8.  -6.  -4.  -2.   0.   2.   4.   6.   8.  10.]
```

[57]:
```
print('Value -1: ', function1_for_y(x), 'Value -2: ',function2_for_y(x))
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Input In [57], in <cell line: 1>()
----> 1 print('Value -1: ', function1_for_y(x), 'Value -2: ',function2_for_y(x)
```

```
Input In [52], in function1_for_y(x)
      1 def function1_for_y(x):
----> 2     if (x > 0):
      3         y = 10.*x - 1
      4     if (x == 0):

ValueError: The truth value of an array with more than one element is ambiguous
 ↪Use a.any() or a.all()
```

[59]:
```
y = np.zeros([11,1])

for i in range(11): # or range(len(y))
    y[i] = function1_for_y(x[i])

    print(i,x[i],y[i])
```

```
0 -10.0 [-4.]
1 -8.0 [-3.2]
2 -6.0 [-2.4]
3 -4.0 [-1.6]
4 -2.0 [-0.8]
5 0.0 [1.]
6 2.0 [19.]
7 4.0 [39.]
8 6.0 [59.]
9 8.0 [79.]
10 10.0 [99.]
```

And/or statements And/or statements help with creating filters and conditions.

[60]:
```
a = 100
b = 50
c = 10

print(a>b, a>c,a==b)
```

```
True True False
```

[61]:
```
if (a>b and b>c):
    print('a is the biggest number.')
```

```
a is the biggest number.
```

[62]:
```
print( (a>b and b>c))
```

```
True
```

[63]:
```
if (b>a and a>c):
    print ('b is the biggest number.')
```

[64]:
```
print( (b>a and a>c))
```

```
False
```

[65]:
```
print( (b<a or b<c))
```

```
True
```

[66]:
```
print(x)
print('Positive x values: ', x[(x>0)])
print('Negative x values: ', x[(x<0)])
```

```
[-10.  -8.  -6.  -4.  -2.   0.   2.   4.   6.   8.  10.]
Positive x values:  [ 2.  4.  6.  8. 10.]
Negative x values:  [-10.  -8.  -6.  -4.  -2.]
```

[68]:
```
print('y values for positive x values: ', y[(x>0)])
print('y values for negative x values: ', y[(x<0)])
```

```
y values for positive x values:  [[19.]
 [39.]
 [59.]
 [79.]
 [99.]]
y values for negative x values:  [[-4. ]
 [-3.2]
 [-2.4]
 [-1.6]
 [-0.8]]
```

[69]:
```
a = [1,2,3,4,5,6,7,8,9,10]
```

[78]:
```
a = np.array(a)

(a.size-1)/2
```

[78]: 4.5

[83]:
```
a.shape()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Input In [83], in <cell line: 1>()
----> 1 a.shape()
```

```
TypeError: 'tuple' object is not callable
```

Congratulations! You have completed the NumPy Workbook. Copy this workbook with a new name and try changing the exercises to explore the functions further.