Content ¶

- **Creating special arrays**
  - `zeros()`
  - `ones()`
  - `diag()`
  - `identity()`
- **Numpy based Mathematical functions**
  - **Absolute values**
    - `np.absolute()`
    - `np.abs()`
  - **Trigonometric Functions**
    - `np.sin()` , `np.cos()`
  - **Exponential and Logarithmic Functions**
    - `np.exp()` , `np.log()` , `np.log2()` , `np.log10()`

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

Creating special arrays using Numpy

How will create numpy array with all zeros?

In [ ]:
```python
np.zeros(3)
# Pass in how many values you need in array
# All values will be zeroes
```

Out[11]:
```
array([0., 0., 0.])
```

- We can also **pass in shape** (as tuple) in case we want to create 2D array

In [ ]:
```python
np.zeros((2, 3))
```

Out[12]:
```
array([[0., 0., 0.],
       [0., 0., 0.]])
```

How will you create numpy array with all ones?

In [ ]:
```
# Just like np.zeroes, but initialize all values to 1
np.ones(3)
```

Out[13]:
```
array([1., 1., 1.])
```

In [ ]:
```
# 2D
np.ones((2,3))
```

Out[14]:
```
array([[1., 1., 1.],
       [1., 1., 1.]])
```

Now, do we need `np.twos()` , `np.threes()` , `np.fours()` , ... `np.hundreds()` ?

- NO
- We can just create array using `np.ones()` and multiply with required value

In [ ]:
```
np.ones((2, 3)) * 5
```

Out[15]:
```
array([[5., 5., 5.],
       [5., 5., 5.]])
```

Notice the datatype of arrays created using `.zeros()` and `.ones()` function

In [ ]:
```
a = np.zeros((2,2))
a
```

Out[16]:
```
array([[0., 0.],
       [0., 0.]])
```

In [ ]:
```
a.dtype
```

Out[3]:
```
dtype('float64')
```

It by defaults creates array with dtype float

One more frequently appearing arrays are diagional matrices

In [ ]:
```
np.diag([1, 2, 3])
# We pass values for diagonal elements as a list
# All other elements are zero
```

Out[17]:
```
array([[1, 0, 0],
       [0, 2, 0],
       [0, 0, 3]])
```

How will you create identity matrix?

First of all, what are identity matrices

- It's a **square matrix**
- Where **all diagonal values are 1**
- **All non-diagonal values are 0**

In [ ]:
```python
np.identity(3)
# Pass in the single dimension of required square identity matrix
```

Out[18]:
```
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
```

Numpy Mathematical functions

Absolute values

At times, we might need to find absolute values of elements in array. Numpy provides a very easy-to-use function for this purpose

`np.absolute()`

It calculate the absolute value element-wise. It returns an ndarray containing the absolute value of each element.

In [ ]:
```python
x = np.array([-1.2, 1.2])
np.absolute(x)
```

Out[3]:
```
array([1.2, 1.2])
```

If the input is a complex value, like `x = a + ib` , the absolute value is $\sqrt{(a^2 + b^2)}$. This is a scalar if x is a scalar.

In [ ]:
```python
np.absolute(1.2 + 1j)
```

Out[4]:
```
1.5620499351813308
```

`np.abs()`

The `abs` function can be used as a shorthand for `np.absolute` on ndarrays.

In [ ]:
```
x = np.array([-1.2, 1.2])
np.abs(x)
```

Out[5]:
```
array([1.2, 1.2])
```

Trigonometric Functions

In addition to arithmetic expressions using operators, Numpy provides functions for element-wise evaluation of many elementary trigonometric functions and operations.

Each of these functions takes a single array (of arbitrary dimension) as input and returns a new array of the same shape, where for each element the function has been applied to the corresponding element in the input array.

`np.sin()` , `np.cos()`

This function takes only one argument and is used to compute the sine function for all values in the array:

In [ ]:
```
x = np.linspace(-1, 1, 11)
x
```

Out[24]:
```
array([-1. , -0.8, -0.6, -0.4, -0.2,  0. ,  0.2,  0.4,  0.6,  0.8,  1. ])
```

In [ ]:
```
y = np.sin(np.pi * x)
```

In [ ]:
```
np.round(y, decimals=4)
```

Out[26]:
```
array([-0.    , -0.5878, -0.9511, -0.9511, -0.5878,  0.    ,  0.5878,
        0.9511,  0.9511,  0.5878,  0.    ])
```

Here we also used the constant `np.pi` and the function `np.round()` to round the values of `y` to four decimals.

Like the `np.sin` function, many of the elementary trigonometric math functions take one input array and produce one output array. We can also make these functions operate on two input arrays and return one array:

For example: $\sin^2 x + \cos^2 x = 1$

In [ ]:
```
np.add(np.sin(x) ** 2, np.cos(x) ** 2)
```

Out[27]:
```
array([1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.])
```

In [ ]:
```
np.sin(x) ** 2 + np.cos(x) ** 2
```

Out[28]:
```
array([1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.])
```

Exponential and Logarithmic Functions

`np.exp()`

This function returns element-wise exponent raised to power of element's value

In [ ]:
```
x = np.arange(0,3)
x
```

Out[33]:
```
array([0, 1, 2])
```

In [ ]:
```
np.exp(x) # returns e**0, e**1, e**2
```

Out[34]:
```
array([1.        , 2.71828183, 7.3890561 ])
```

`np.log()` , `np.log2()` , `np.log10()`

These functions return Logarithms of base e, 2, and 10, respectively.

In [ ]:
```
x = np.arange(1,11)
x
```

Out[43]:
```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
```

In [ ]:
```
np.log(x)
```

Out[44]:
```
array([0.        , 0.69314718, 1.09861229, 1.38629436, 1.60943791,
       1.79175947, 1.94591015, 2.07944154, 2.19722458, 2.30258509])
```

In [ ]:
```
np.log10(x)
```

Out[45]:
```
array([0.        , 0.30103   , 0.47712125, 0.60205999, 0.69897   ,
       0.77815125, 0.84509804, 0.90308999, 0.95424251, 1.        ])
```
