# ENGG1811 Computing for Engineers

## Week 7B: numpy elementwise arithmetic operations

# Topic to be covered

- Elementwise arithmetic operations

# Arithmetic operators

- You can use +, -, *, /, ** on two numpy arrays
  - They perform elementwise operations
  - See the next two slides for illustration
- The shapes of these arrays are required to be **compatible**.
- We will first consider the case where both arrays have the same shape
  - Code in numpy_arith_1.py

# Elementwise multiplication

```
array1 = np.array([ [-3.2,  0,  0.5, 5.8],
                    [   6, -4,  6.2, 7.1],
                    [ 3.8,  5,  2.7, 3.7]])

array2 = np.array([ [-1.2,  2, -3.1, 0.0],
                    [   4, -5,  3.5, 7.1],
                    [ 2.7,  2,  1.7, 3.4]])
```

```
array_mul = array1 * array2 # NOT matrix multiplication

array([[ 3.84,  0.  , -1.55,  0.  ],
       [24.  , 20.  , 21.7 , 50.41],
       [10.26, 10.  ,  4.59, 12.58]])
```

# Elementwise division

```
array1 = np.array([ [-3.2,  0,  0.5, 5.8],
                    [   6, -4,  6.2, 7.1],
                    [ 3.8,  5,  2.7, 3.7]])

array2 = np.array([ [-1.2,  2, -3.1, 0.0],
                    [   4, -5,  3.5, 7.1],
                    [ 2.7,  2,  1.7, 3.4]])
```

```
array_div = array1 / array2

array([[ 2.667,  0.        , -0.161,         inf],
       [ 1.5  ,  0.8       ,  1.771,          1.],
       [ 1.407,  2.5       ,    1.8,       1.088]])
```

# Example on using elementwise arithmetic operations (1)

- You work in a company and every day you take product samples to determine their quality. The results on Monday were:
  - 16 devices passed
  - 4 devices failed
- You can calculate the percentage of devices passing the test by

```
# %% For Monday

# The quality check results for Monday
num_devices_passed = 16
num_devices_failed = 4

# Percentages of devices passed
percentage_passed = \
      num_devices_passed / (num_devices_passed + num_devices_failed)
```

calculates

$$\frac{16}{16+4}$$

numpy_arith_1_example.py

# Example on using elementwise arithmetic operations (2)

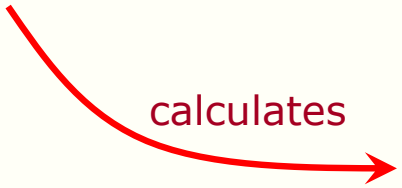
- You store the test results for Monday to Wednesday in two arrays

| | P | F |
|---|---|---|
| M | 16 | 4 |
| T | 28 | 2 |
| W | 35 | 5 |

```
num_devices_passed = np.array([16, 28, 35])
num_devices_failed = np.array([4, 2, 5])
```

- You can compute the percentages of devices passing the tests over Mon-Wed by:

```
percentage_passed = \
num_devices_passed / (num_devices_passed + num_devices_failed)
```




$$\left[ \frac{16}{16+4}, \frac{28}{28+2}, \frac{35}{35+5} \right]$$

# Discussion

- Observation: We can use **the same** Python expression for scalar and array computations

```
percentage_passed = \
num_devices_passed / (num_devices_passed + num_devices_failed)
```

- That's why elementwise computation is useful!
- However, some method of storing data will make using elementwise computation difficult

| | P | F |
|---|---|---|
| M | 16 | 4 |
| T | 28 | 2 |
| W | 35 | 5 |

```
Works:
num_devices_passed = np.array([16, 28, 35])
num_devices_failed = np.array([4, 2, 5])
```

```
Does not work: [16, 4], [28, 2], [35, 5]
```

- Forum exercise: Use 2-D array

# More on numpy arithmetic operators

- You have seen that you can use the numpy arithmetic operators on two arrays of the same shape
- You can also use the numpy arithmetic operators on two arrays when
  - One array is a scalar
  - The other is a numpy array of any shape
- Let us look at the examples in numpy_arith_2.py

# Elementwise division: an array and a scalar

```
array1 = np.array([ [-3.2,  1,  0.5, 5.8],
                    [   6, -4,  6.2, 7.1],
                    [ 3.8,  5,  2.7, 3.7]])
```

```
array_div_1 = array1 / 2.0
array([[-1.6 ,  0.5 ,  0.25,  2.9 ],
       [ 3.  , -2.  ,  3.1 ,  3.55],
       [ 1.9 ,  2.5 ,  1.35,  1.85]])
```

```
array_div_2 = 2.0 / array1
array([[-0.625,  2.,      4.,  0.345],
       [  0.333, -0.5,  0.322,  0.282],
       [  0.526,  0.4,  0.741,  0.541]])
```

# Exercise: Obtaining an array from another array

- If you drop an object from a height of *h0* and if the air resistance is small, then the height of the object at time *t* is

$$h0 - 0.5 * g * t^2$$

where g is the acceleration due to gravity

- Assume g = 9.81. Let h0 = 1000.
- Given: time_array = np.array([0, 2, 4, 6, 8])

- Determine the height of the objects at the time instants in time array and store the results in an array
  - Hint: Next page

# Exercise: Hint

File: numpy_arith_2_prelim.py

- The following hint for array [0, 2, 4]

Start from

$$[0 \quad , 2 \quad , 4 \quad ]$$

$$= h0 - 0.5 * g * [\, 0^2 \quad , 2^2 \, , \quad 4^2]$$

Keep working backwards until you use [0,2,4]

$$= h0 - [\, 0.5 * g * 0^2 \quad , 0.5 * g * 2^2 \, , 0.5 * g * 4^2]$$

Work backwards

$$= [\, h0 - 0.5 * g * 0^2 \quad , h0 - 0.5 * g * 2^2 \quad , h0 - 0.5 * g * 4^2]$$

final result wanted

$$= [1000. \quad , \; 980.38. \quad , \; 921.52]$$

# Mathematical functions

- The numpy mathematical functions are documented here:
  - https://docs.scipy.org/doc/numpy/reference/routines.math.html
- Example: sin, cos, asin, log, exp, sqrt, absolute

- Notes:
  - You need to append the library name, say you import numpy as np, then np.cos etc.
  - They are different to those in the math library
  - They are **elementwise operation**. The output is an array of the same size as input and the operation is applied to each element (illustrated on the next slide)
- Code in numpy_math_func.py

# Elementwise operation

```
array2 = np.array([[-1.2,  2. , -3.1,  4.5],
                   [ 4. , -5. ,  3.5,  7.1],
                   [ 2.7,  9. ,  1.7,  3.4]])

array2_sin = np.sin(array2)
array([[-0.93203909,  0.90929743, -0.04158066, -0.97753012],
       [-0.7568025 ,  0.95892427, -0.35078323,  0.72896904],
       [ 0.42737988,  0.41211849,  0.99166481, -0.2555411 ]])
```

sin(4.5)

sin(2.7)

sin(1.7)

# Summary

- Numpy elementwise operations
- Main application:
  - To produce a new array from the given arrays
- Elementwise operations allow you to use the same Python expression for scalars as well as for arrays
- You used loops to create a new list from an existing list. In numpy, loops are not necessary.