

# CLASS 12: MATHEMATICAL OPERATIONS IN PYTHON

ENGR 102 – Introduction to Engineering

## 2

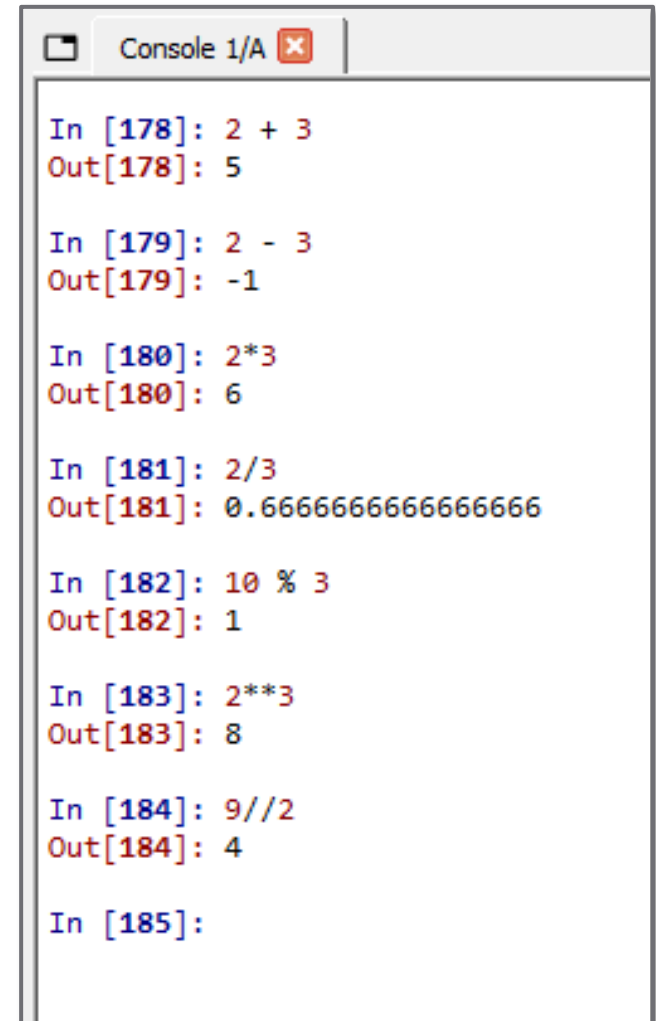
# Mathematical Operations

Python includes the most basic mathematical operations. Other math functions will be accessed by importing the NumPy package

# Basic Mathematical Operations

3

- Python itself includes only seven mathematical operators
  - ▣ Addition: +
  - ▣ Subtraction: -
  - ▣ Multiplication: \*
  - ▣ Division: /
  - ▣ Modulus: %
  - ▣ Exponentiation: \*\*
  - ▣ Floor division: //



```
Console 1/A x
In [178]: 2 + 3
Out[178]: 5

In [179]: 2 - 3
Out[179]: -1

In [180]: 2*3
Out[180]: 6

In [181]: 2/3
Out[181]: 0.6666666666666666

In [182]: 10 % 3
Out[182]: 1

In [183]: 2**3
Out[183]: 8

In [184]: 9//2
Out[184]: 4

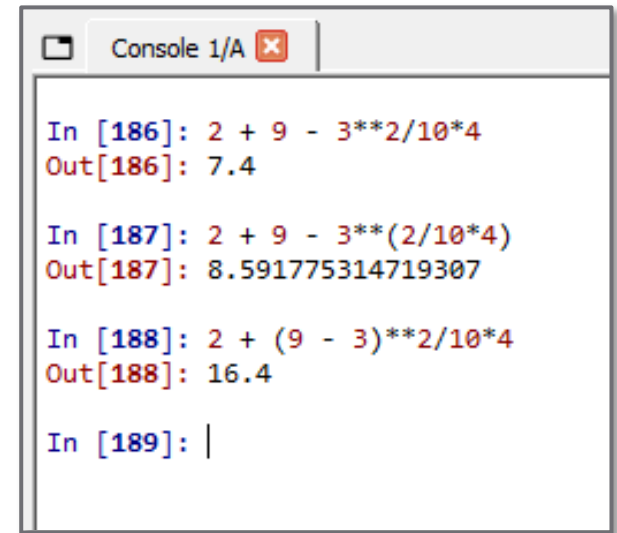
In [185]:
```

# Order of Operations

4

## □ Python order of operations:

- 1) ( ) parentheses
- 2) ^ exponentiation
- 3) - negation
- 4) \*, / multiplication, division
- 5) +, - addition, subtraction



```
Console 1/A x  
In [186]: 2 + 9 - 3**2/10*4  
Out[186]: 7.4  
In [187]: 2 + 9 - 3**(2/10*4)  
Out[187]: 8.591775314719307  
In [188]: 2 + (9 - 3)**2/10*4  
Out[188]: 16.4  
In [189]: |
```

- ## □ Expressions are evaluated left to right within each level of the precedence hierarchy

# Other Built-In Python Functions

5

- A few other math-related built-in Python functions:

- ▣ `abs(x)`: absolute value

```
>>> a = abs(-1.76)
```

```
1.76
```

```
>>> z = abs(2 - 2j)
```

```
2.828
```

- ▣ `len(x)`: returns the length of an object

```
>>> len([2, 4, 5, 3, 1])
```

```
5
```

```
>>> len('Hello, World!')
```

```
13
```

# Other Built-In Python Functions

6

- A few other math-related built-in Python functions:

- `max(x)`: maximum value in a sequence

```
>>> x_max = max([2, 4, 5, 3, 1])
```

```
5
```

- `min(x)`: minimum value in a sequence

```
>>> x_min = min([2, 4, 5, 3, 1])
```

```
1
```

- `type(x)`: returns the type of an object

```
>>> type([2, 4, 5, 3, 1])
```

```
list
```

```
>>> type('Hello, World!')
```

```
str
```

# NumPy

Here we will introduce the concept of *packages* **and** will look specifically at the package we will use most for mathematical operations, *NumPy*.

# Packages

8

## □ *Python packages*

- Libraries consisting of multiple *modules*, or individual Python files
- Modules within a package define
  - Data types
  - Functions
- Must install a package before we can use it
  - Anaconda distribution includes all the packages we will need
- Must import a package in our code before we can use it
  - Use the `import` function

## □ Packages available for

- Array processing and mathematics
- Plotting
- Data analysis
- GUI development
- Much, much more ...



# NumPy

9

- We will use the NumPy (**N**umerical **P**ython) package extensively
- Fundamental data type:
  - ▣ Multi-dimensional array object – ndarray
    - Useful for engineering computation
- Many built-in functions
  - ▣ Mathematical operations, e.g.:
    - Trigonometric functions
    - Exponents and logarithms
    - Complex number operations
  - ▣ Array creation and manipulation routines
  - ▣ Polynomial creation, manipulation, fitting, etc.
  - ▣ Much more ...



# Using NumPy

10

- To use NumPy functions and data types, we must first ***import*** it:

```
>>> import numpy as np
```

- ▣ We can assign it a shortened name, np, to keep our code clean
- To call functions defined in NumPy, ***precede the function name with np.***

```
>>> N = np.log2(1024)
```

```
>>> x = 3*np.sin(np.pi/2)
```

- We'll now introduce a small sample of NumPy functions

# NumPy – Trigonometric Functions

11

- `sin(x)`, `cos(x)`, `tan(x)`

- ▣ Input in radians

```
>>> y = np.sin(x)
```

```
>>> y = np.sin(np.radians(x))
```

- `arcsin(x)`, `arccos(x)`, `arctan(x)`

- ▣ Inverse trig functions

- ▣ Output in radians

```
>>> theta = np.arcsin(0.6)
```

# NumPy – Trigonometric Functions

12

- `arctan2(x)` – quadrant-aware inverse tangent
  - ▣ Accounts for the difference between, e.g. ,  $45^\circ$  and  $225^\circ$
  - ▣ Output in radians

```
>>> phi = np.arctan2(-4, 3)
```

```
>>> phi_deg = np.degrees(np.arctan2(-4, 3))
```

- `degrees(x)` – converts from radians to degrees

```
>>> ang45 = np.degrees(np.pi/4)
```

- `radians(x)` – converts from degrees to radians

```
>>> angPi = np.radians(180)
```

# NumPy – Rounding

13

- `around(x, decimals=0)` – round to the specified number of decimals (default, 0)

```
>>> xint = np.around(1.6)
2.0
```

```
>>> xrnd = np.around(np.pi, decimals=2)
3.14
```

- Numbers exactly halfway between rounded decimal values round to the nearest ***even value***

```
>>> x0 = np.around(2.5)
2.0
```

```
>>> x1 = np.around(1.65, decimals=1)
1.6
```

```
>>> y1 = np.around(1.55, decimals=1)
1.6
```

# NumPy – Rounding

14

- `fix(x)` – round to the nearest integer ***toward zero***

```
>>> xfix = np.fix(1.2)
```

```
1.0
```

```
>>> yfix = np.fix(-2.8)
```

```
-2.0
```

- `floor(x)` – round to the nearest integer ***toward negative infinity***

```
>>> xfloor = np.floor(1.6)
```

```
1.0
```

```
>>> xflr = np.floor(-1.2)
```

```
-2.0
```

- `ceil(x)` – round to the nearest integer ***toward positive infinity***

```
>>> xceil = np.fix(1.2)
```

```
2.0
```

```
>>> yceil = np.fix(-2.8)
```

```
-2.0
```

# NumPy – Exponents

15

- `exp(x)` – exponential:  $e^x$

```
>>> y = np.exp(4.1)
60.3403
```

```
>>> e = np.exp(1)
2.71828
```

- `exp2(x)` – power of 2:  $2^x$

```
>>> x = np.exp2(3)
8.0
```

```
>>> N = np.exp2(10)
1024.0
```

# NumPy – Logarithms

16

- $\log(x)$  – natural log

```
>>> y = np.log(5)  
1.609
```

- $\log_{10}(x)$  – base-10 logarithm

```
>>> x = np.log10(1e4)  
4.0
```

- $\log_2(x)$  – base-2 logarithm

```
>>> x = np.log2(256)  
8.0
```



# NumPy – Complex Numbers

17

- `real(z)` – real part of a complex number

```
>>> x = np.real(3 + 5j)
3.0
```

- `imag(z)` – imaginary part of a complex number

```
>>> y = np.imag(3 + 5j)
5.0
```

- `angle(z)` – angle of complex number in radians

```
>>> x = np.degrees(np.angle(2 + 2j))
45
```

- `conj(z)` – complex conjugate

```
>>> x = np.conj(3 + 5j)
3 - 5j
```

# NumPy – Miscellaneous

18

- `sqrt(x)` – square root

```
>>> y = np.sqrt(2)
1.4142
```

- `sum(x)` – sum of all elements in a sequence

```
>>> total = np.sum([2, 4, 5, 3, 1])
15
```

- `sign(x)` – returns: -1 if  $x < 0$ , 0 if  $x == 0$ , 1 if  $x > 0$

```
>>> np.sign([-12, 4, 6, 0, -3])
array([-1, 1, 1, 0, -1])
```

# NumPy – Element-Wise Operations

19

- Numpy functions operate element-by-element on array (or other sequence) inputs

- Return **array** outputs (more later)

```
>>> np.log10([1e4, 0.001, 10, 1e-6])  
array([ 4., -3.,  1., -6.] )
```

```
>>> np.sqrt([4, 9, 25, 1e4])  
array([ 2.,  3.,  5., 100.] )
```

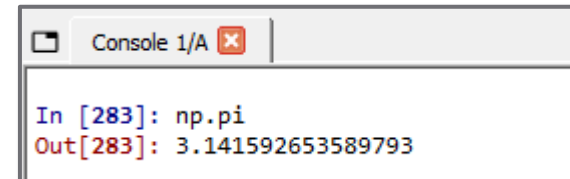
- Eliminates the need to explicitly perform the operation on each element in an array

# Built-In Constants

20

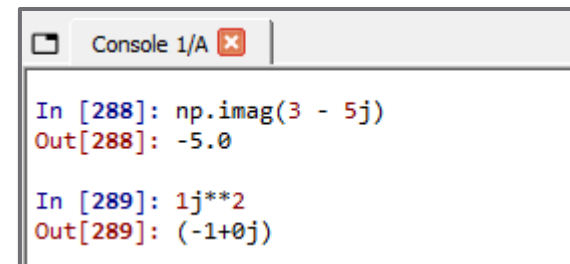
## □ Some built-in Python and Numpy constants:

□  $\pi$ : `np.pi`



```
Console 1/A ✕  
In [283]: np.pi  
Out[283]: 3.141592653589793
```

□ Imaginary unit ( $\sqrt{-1}$ ): `j`



```
Console 1/A ✕  
In [288]: np.imag(3 - 5j)  
Out[288]: -5.0  
In [289]: 1j**2  
Out[289]: (-1+0j)
```

□ Infinity ( $\infty$ ): `inf`

□ Not-a-number: `NaN` or `nan`

- Both `inf` and `nan` often result from algorithmic errors

# Math in Python and NumPy

21

## Exercise

- Use Python and NumPy to calculate each of the following expressions

- $\left[ e^{\left( \frac{-0.4\pi}{\sqrt{1-0.4^2}} \right)} \right] \cdot 100$

- $\frac{-\ln\left(\frac{15}{100}\right)}{\sqrt{\pi^2 + \ln^2\left(\frac{15}{100}\right)}}$

- $\frac{12}{3^2} \left[ 1 - e^{-0.1 \cdot 8} \cos(2\pi \cdot 12 \cdot 8) \right]$