ESCI 386 – Scientific Programming, Analysis and Visualization with Python

Lesson 4 – Operators and Mathematical Functions
Numerical Operators

• + addition

• – subtraction

• * multiplication

• / division
Numerical Operators (cont.)

• // truncating division
  16.3 // 5.2 => 3.0

• ** power

• % modulo (returns the remainder)
  5 % 3 => 2
  5.2 % 3 => 2.2
Comparison Operators

• < less than
• > greater than
• == equal to
• != not equal to
• >= greater than or equal to
• <= less than or equal to
Augmented Assignment Operators

• The augmented assignment operators are shorthand operators and take the form `x += y`, which is the same as `x = x + y`.

• This works not only with addition (+), but also with subtraction, multiplication, division, truncated division, and powers.
Augmented Assignment Examples

```python
>>> x = 5
>>> x += 2
>>> x
7
>>> x *= 8
>>> x
56
>>> x /= 9.0
>>> x
6.222222222222222
>>> x **= 3
>>> x
240.89986282578877
```
Boolean Operators

- The three Boolean operators in Python are `or`, `and`, and `not`.

```python
>>> x, y, z = True, False, True
>>> x or y
True
>>> x or z
True
>>> y or z
True
>>> x and y
False
>>> x and z
True
>>> y and z
False
>>> not x
False
>>> not y
True
>>> not z
False
```
Mathematical Functions

• Python has a limited number of built-in mathematical functions:
  – abs(x) absolute value of x
  – divmod(x,y) returns a tuple with (x // y, x % y)
  – pow(x,y) same as $x^{**y}$
  – round(x, [m]) rounds x to nearest integer value, unless optional integer m is given, in which case it round to nearest multiple of $10^{-m}$. 
Built-in Math Functions Examples

>>> round(-2.4)
-2.0
>>> round(-2.6)
-3.0
>>> round(-2.6473, 2)
-2.65
>>> divmod(12, 8)
(1, 4)
The math Module

• More advanced mathematical functions are contained in the math module, which must be imported before use, either as
  import math
  — or
  import math as ma
The NumPy module contains many of the same mathematical functions as the math module, and is the preferred module to use for math.

NumPy must be imported before use, either as

```python
import numpy
```

or

```python
import numpy as np
```
Constants in math Module

• $pi$ returns the value of $pi$

• $e$ returns the value of $e$
Functions in the math Module

• `pow(x,y)` same as `x**y`.
  – `pow(1.0,x)` and `pow(x, 0.0)` always return 1.0, even if `x` is zero or NaN.

• `sqrt(x)` returns the square root of `x`. 
Trig Functions in the numpy Module

- `arccos(x)` arc cosine
- `arccosh(x)` inverse cosh
- `arcsin(x)` arc sin
- `arcsinh(x)` inverse sinh
- `arctan(x)` arc tangent
- `arctan2(y,x)` – arc tangent of \( y/x \).
- `arctanh(x)` inverse tanh
- `cos(x)` cosine of \( x \)
- `cosh(x)` hyperbolic cosine
- `degrees(x)` – converts \( x \) to degrees from radians
- `radians(x)` – converts \( x \) to radians from degrees
- `sin(x)` sin of \( x \)
- `sinh(x)` hyperbolic sine
- `tan(x)` tangent of \( x \)
- `tanh(x)` hyperbolic tangent

Note: All angles are in radians unless otherwise specified!
Exponents and Logs in the math Module

- \( \exp(x) \) returns \( e^x \)
- \( \expm1(x) \) returns \( e^x - 1 \).
  - This is more accurate than \( \exp(x)-1 \) for small values of \( x \).
- \( \log(x) \) returns \( \ln x \).
- \( \log(x, n) \) returns \( \log_n x \).
- \( \log1p(x) \) returns \( 1+\ln x \).
  - This is more accurate than \( \log(x) + 1 \) for small values of \( x \).
- \( \log10(x) \) returns the base 10 log of \( x \)
  - preferred over \( \log(x,10) \).
Numeric Functions in the numpy Module

- `ceil(x)` the smallest integer \(\geq\) to \(x\).
- `copysign(x, y)` returns \(x\) with the same sign as \(y\).
- `floor(x)` the largest integer \(\geq\) \(x\).
- `fabs(x)` absolute value.
- `trunc(x)` truncates \(x\) to integer.
Modulo Arithmetic in numpy Module

• $\text{fmod}(x, y)$ like $x \% y$
  – Use $\text{fmod}(x, y)$ if either $x$ or $y$ are floating point values
  – Use $x \% y$ if both $x$ and $y$ are integers.

• $\text{modf}(x)$ breaks $x$ into integer and fractional parts
  – $\text{np.modf}(89.4357) \Rightarrow (0.4356999999999971, 89.0)$
Special Functions in numpy Module

- `erf(x)` error function
- `erfc(x)` complementary error function
- `factorial(x)` returns $x!$
- `gamma(x)` gamma function
- `lgamma(x)` returns $\log(\text{gamma}(x))$
- `hypot(x, y)` length of hypotenuse of right triangle with sides $x$ and $y$. 
Functions to Check for Infinity/NaN in the numpy Module

- `isinf(x)` returns True if `x` is positive or negative infinity.

- `isnan(x)` returns True if `x` is not a number (NaN).
Loading Modules

• All of methods and attributes of a module can be loaded by simply using the import command.

• We then access the functions and constants by prefacing them with numpy.

```python
>>> import numpy
>>> numpy.cos(0.5)
0.87758256189037276
```
Aliasing Modules

• We can use an alias when importing a module.
  – Avoids having to repeatedly typing long module names

>>> import numpy as np
>>> np.cos(0.5)
0.87758256189037276
Importing Individual Functions from Modules

- We can import individual functions or constants from modules.
  - They can also be aliased on import

```python
>>> from numpy import cos
>>> cos(0.5)
0.87758256189037276
>>> from math import cos as macos
>>> macos(0.5)
0.8775825618903728
```

Note that `numpy.cos()` is not the same as `math.cos()`!
Don’t Import All Functions Using *!

- We can import every function and constant and then not have to preface them with the module name

- However, you should avoid this!

- Can cause confusion if multiple modules have functions with the same name.
Example of What Not to Do!

```python
>>> from numpy import *
>>> from math import *
>>> cos(0.5)
0.8775825618903728
```

Which `cos()` function is this? Is it from numpy or from math? It’s from whichever one was loaded last, but it can get confusing, so avoid importing using the *! Don’t do this!