1 Accessing math functions through the math module and accessing help

Here we are going to use the math module as a an untroduction to using modules. The math module contains a range of useful mathemetical functions that are not built into Python directly. So let's go ahead and start by importing the module. Module imports are usually performed at th start of a programme.

```
import math
```

When this type of import is used Python loads up a link to the module so that module functions and variables may be used. Here for example we access the value of pi through the module.

```
print (math.pi)

OUT:
3.141592653589793
```

Another way of accessing module contents is to directly load up a function ror a variable into Python. When we do this we no longer need to use the module name after the import. This method is not generally recommended as it can lead to conflicts of names, and is not so clear where that function or variable comes from. But here is how it is done.

```
from math import pi
print (pi)

OUT:
3.141592653589793
```

Multiple methods and variables may be loaded at the same time in this way.

```
from math import pi, tau, log10
print (tau)
print (log10(100))

OUT:
6.283185307179586
2.0
```

But usually it is better practice to keep using the the library name in the code.

```
print (math.log10(100))

OUT:
2.0
```

To access help on any Python function use the help command in a Python interpreter.

```
help (math.log10)

Help on built-in function log10 in module math:

log10(...)
    log10(x)

    Return the base 10 logarithm of x.
```

To find out all the methods in a module, and how to use those methods we can simply type help (module_name) into the Python interpreter. The module must first have been imported, as we did for math above.

```
help (math)

OUT:
```

```
Help on module math:

NAME
    math

MODULE REFERENCE
    https://docs.python.org/3.6/library/math

    The following documentation is automatically generated from the Python
    source files.  It may be incomplete, incorrect or include features that
    are considered implementation detail and may vary between Python
    implementations.  When in doubt, consult the module reference at the
    location listed above.

DESCRIPTION
    This module is always available.  It provides access to the
    mathematical functions defined by the C standard.

FUNCTIONS
    acos(...)
        acos(x)

        Return the arc cosine (measured in radians) of x.

    acosh(...)
        acosh(x)

        Return the inverse hyperbolic cosine of x.

    asin(...)
        asin(x)

        Return the arc sine (measured in radians) of x.

    asinh(...)
        asinh(x)

        Return the inverse hyperbolic sine of x.

    atan(...)
        atan(x)

        Return the arc tangent (measured in radians) of x.

    atan2(...)
        atan2(y, x)

        Return the arc tangent (measured in radians) of y/x.
        Unlike atan(y/x), the signs of both x and y are considered.

    atanh(...)
        atanh(x)

        Return the inverse hyperbolic tangent of x.

    ceil(...)
        ceil(x)
```

```
        Return the ceiling of x as an Integral.
        This is the smallest integer >= x.

    copysign(...)
        copysign(x, y)

        Return a float with the magnitude (absolute value) of x but the sign
        of y. On platforms that support signed zeros, copysign(1.0, -0.0)
        returns -1.0.

    cos(...)
        cos(x)

        Return the cosine of x (measured in radians).

    cosh(...)
        cosh(x)

        Return the hyperbolic cosine of x.

    degrees(...)
        degrees(x)

        Convert angle x from radians to degrees.

    erf(...)
        erf(x)

        Error function at x.

    erfc(...)
        erfc(x)

        Complementary error function at x.

    exp(...)
        exp(x)

        Return e raised to the power of x.

    expm1(...)
        expm1(x)

        Return exp(x)-1.
        This function avoids the loss of precision involved in the direct evaluation of
exp(x)-1 for small x.

    fabs(...)
        fabs(x)

        Return the absolute value of the float x.

    factorial(...)
        factorial(x) -> Integral

        Find x!. Raise a ValueError if x is negative or non-integral.

    floor(...)
```

```
    floor(x)

    Return the floor of x as an Integral.
    This is the largest integer <= x.

fmod(...)
    fmod(x, y)

    Return fmod(x, y), according to platform C.  x % y may differ.

frexp(...)
    frexp(x)

    Return the mantissa and exponent of x, as pair (m, e).
    m is a float and e is an int, such that x = m * 2.**e.
    If x is 0, m and e are both 0.  Else 0.5 <= abs(m) < 1.0.

fsum(...)
    fsum(iterable)

    Return an accurate floating point sum of values in the iterable.
    Assumes IEEE-754 floating point arithmetic.

gamma(...)
    gamma(x)

    Gamma function at x.

gcd(...)
    gcd(x, y) -> int
    greatest common divisor of x and y

hypot(...)
    hypot(x, y)

    Return the Euclidean distance, sqrt(x*x + y*y).

isclose(...)
    isclose(a, b, *, rel_tol=1e-09, abs_tol=0.0) -> bool

    Determine whether two floating point numbers are close in value.

       rel_tol
           maximum difference for being considered "close", relative to the
           magnitude of the input values
        abs_tol
           maximum difference for being considered "close", regardless of the
           magnitude of the input values

    Return True if a is close in value to b, and False otherwise.

    For the values to be considered close, the difference between them
    must be smaller than at least one of the tolerances.

    -inf, inf and NaN behave similarly to the IEEE 754 Standard.  That
    is, NaN is not close to anything, even itself.  inf and -inf are
    only close to themselves.
```

```
isfinite(...)
    isfinite(x) -> bool

    Return True if x is neither an infinity nor a NaN, and False otherwise.

isinf(...)
    isinf(x) -> bool

    Return True if x is a positive or negative infinity, and False otherwise.

isnan(...)
    isnan(x) -> bool

    Return True if x is a NaN (not a number), and False otherwise.

ldexp(...)
    ldexp(x, i)

    Return x * (2**i).

lgamma(...)
    lgamma(x)

    Natural logarithm of absolute value of Gamma function at x.

log(...)
    log(x[, base])

    Return the logarithm of x to the given base.
    If the base not specified, returns the natural logarithm (base e) of x.

log10(...)
    log10(x)

    Return the base 10 logarithm of x.

log1p(...)
    log1p(x)

    Return the natural logarithm of 1+x (base e).
    The result is computed in a way which is accurate for x near zero.

log2(...)
    log2(x)

    Return the base 2 logarithm of x.

modf(...)
    modf(x)

    Return the fractional and integer parts of x.  Both results carry the sign
    of x and are floats.

pow(...)
    pow(x, y)

    Return x**y (x to the power of y).
```

```
    radians(...)
        radians(x)

        Convert angle x from degrees to radians.

    sin(...)
        sin(x)

        Return the sine of x (measured in radians).

    sinh(...)
        sinh(x)

        Return the hyperbolic sine of x.

    sqrt(...)
        sqrt(x)

        Return the square root of x.

    tan(...)
        tan(x)

        Return the tangent of x (measured in radians).

    tanh(...)
        tanh(x)

        Return the hyperbolic tangent of x.

    trunc(...)
        trunc(x:Real) -> Integral

        Truncates x to the nearest Integral toward 0. Uses the __trunc__ magic method.

DATA
    e = 2.718281828459045
    inf = inf
    nan = nan
    pi = 3.141592653589793
    tau = 6.283185307179586

FILE
    /home/michael/anaconda3/lib/python3.6/lib-dynload/math.cpython-36m-x86_64-linux-gnu.so
```

So now, for example, we know that to take a square root of a number we can use the math module, and use the sqrt() function, or use the pow() function which can do any power or root.

```
print (math.sqrt(4))
print (math.pow(4,0.5))

OUT:
2.0
2.0
```

In Python you might read about packages as well as modules. The two names are sometimes used interchangeably, but strictly a package is a collection of modules.