ICS3U: Additional Mathematical Functions

Built-In Mathematical Functions

In addition to mathematical operators like addition and exponentiation, Python also contains several built-in mathematical **functions** that can be *called* to calculate a value. To call a function in Python, write the name of the function, followed by a set of round brackets that contain one or more **arguments** (values). For example, to calculate the absolute value of a number, you can use the built-in `abs` function.

```
>>> abs(-5)
5
```

Another useful function is for rounding a value to a specified number of decimal places. This requires two arguments: the value to be rounded, and the number of decimal places to use. This function is called `round`.

```
>>> round(12.34567, 3)
12.346
```

Other handy functions are `min` and `max`. These two functions take a sequence of values, and return the smallest and largest values respectively.

```
>>> min(3, 1, 7, 5)
1
>>> max(3, 1, 7, 5)
7
```

As we progress through this course, we will introduce additional functions and revisit other uses of some of these mathematical functions.

Functions From the `math` Module

Let's say that we want to calculate the square root of 42. We know that the answer is somewhere between 6 and 7, since $6^2=36$ and $7^2=49$, but we would like to know the answer to a few decimal places. You might try typing something like the following.

```
>>> sqrt(42)
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    sqrt(42)
NameError: name 'sqrt' is not defined
```

The run-time error states that there is no function named `sqrt`; however, it *does* exist. The problem is that `sqrt` is not a built-in function like `abs` or `round`. In an effort to conserve memory, Python only loads a small number of functions at start-up. All other functions are stored in **modules**, which are just Python files that can be **imported** into your program.

To import the `math` module, use the `import` command.

```
>>> import math
```

It might seem like nothing has happened; however, Python has loaded all of the functions contained in the `math` module into memory, so that they are available for you to use. If there was an error, say from trying to load a module that does not exist, you would see an error as follows.

```
>>> import ABC
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    import ThisModuleDoesNotExist
ImportError: No module named ABC
```

This is not the case, however, so we know that the module was found. Unfortunately, if we try to calculate the square root of 42, it still causes an error.

```
>>> sqrt(42)
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    sqrt(42)
NameError: name 'sqrt' is not defined
```

To reference a function that has been imported, we must *prefix* (attach to the front) the function with the name of the module in which it is stored. This points Python toward the correct file. This might seem like a strange requirement, since we imported the module and Python should know where it is, but there are practical reasons for this decision.

```
>>> math.sqrt(42)
6.48074069840786
```

Finally, we get the answer we expected. Let's round that two two decimals so that it is easier to read.

```
>>> round(math.sqrt(42), 2)
6.48
```

In the last example, we **nested** a function inside of another one. The value produced by the `sqrt` function was used as an argument in the `round` function. This is very common in computer programs, especially in Python. Instead of writing multiple commands across several lines, we can often combine the commands into a single unified instruction.

Note that the square root function always returns a decimal answer, even if the result is an integer.

```
>>> math.sqrt(9)
3.0
```

In addition to mathematical functions, the `math` module contains various constants too, such as π (pi) and *e* (Euler's number). Both constants are irrational numbers, so their values are only approximations.

```
>>> math.pi
3.141592653589793
>>> math.e
2.718281828459045
```

Trigonometric functions are also found in the `math` module; however, it is important to note that these functions operate with radians, rather than degrees. For example, $\sin 30^{\circ} = 0.5$, but attempting this in Python gives the following.

```
>>> math.sin(30)
-0.9880316240928618
```

To convert an angle from degrees to radians, use the `radians` function.

```
>>> math.radians(30)
0.5235987755982988
>>> math.sin(0.5235987755982988)
0.49999999999999994
```

It is not very efficient to have to cut-and-paste the decimal value 0.52..., so a better idea is to nest the functions like we did earlier.

```
>>> math.sin(math.radians(30))
0.49999999999999994
```

Note that the value is not 0.5 is not *quite* what we expected it to be. This is because of the way in which computers represent decimal values. They are not always precise, and are often subject to rounding errors. For this reason, many financial institutions have laws in place that restrict or prohibit the use or decimal values for calculation purposes.

Inverse trigonometric functions are possible as well. Sine inverse, often denoted as $\sin^{-1}$ on calculators, uses the older terminology arcsine, abbreviated in Python as `asin`. Again, note the slight inaccuracy.

```
>>> math.degrees(math.asin(0.5))
30.000000000000004
```

Logarithmic and exponential calculations can also be made using the functions in the `math` module. For logarithms with a base of 10, use `log10`. For a specified base, you can use `log` instead. e^x can be computed using `exp`.

```
>>> math.log10(100)
2.0
>>> math.log(8, 2)
3.0
```

Some other useful functions, dealing with integers, are the `floor` and `ceil` functions, which round down and up respectively. These are typically denoted $\lfloor x \rfloor$ and $\lceil x \rceil$. Another useful function is for *factorials*, which arise in many counting problems. n factorial is defined as $n! = n \times (n-1) \times (n-2) \times \ldots \times 2 \times 1$.

```
>>> math.floor(4.9)
4
>>> math.ceil(3.1)
4
>>> math.factorial(4)
24
>>> 4*3*2*1
24
```