1 Function calls

A **function** is a named sequence of statements that performs a computation. Later, we will learn how to define our own functions. For now, we learn how to **call** functions that were written by another programmer.

Function calls have the following form:

function_name(arguments)

How python executes a function call:

1. Evaluate the arguments.
2. Call the function, passing in the argument values.
3. Perform the computation of the function and return the resulting value.

Terminology:

- **argument**: a value given to a function
- **pass**: to provide to a function
- **call**: to ask python to evaluate a function
- **return**: to pass back a value (the result of the function's evaluation)

In this example, the function `pow` is *called*. The *arguments* `3` and `4` are *passed* to `pow`. The function `pow` computes three raised to the power of four and *returns* the value 81.

```
>>> pow(3, 4)
81
```

1.1 Built-in functions

```
>>> int(2.3)   # makes a new int from argument, if possible
2
>>> float(2)   # makes a new float from argument, if possible
2.0
>>> x = int(4.3)
>>> type(x)    # returns the argument's type
<type 'int'>
>>> max(-30, 12, 4.0, 6.75)  # returns largest argument
12
>>> min(-30, 12, 4.0, 6.75)  # returns smallest argument
-30
>>> abs(-30)   # returns absolute value of argument
30
```

1.2 Help function

The help function takes one argument, such as the name of a function, and displays documentation.

```
>>> help(abs)
Help on built-in function abs in module __builtin__:

abs(...)
    abs(number) -> number
    Return the absolute value of the argument.
```

In the help documentation, arguments in brackets, such as `z` below, are optional.

```
>>> help(pow)
Help on built-in function pow in module __builtin__:

pow(...)
    pow(x, y[, z]) -> number
    With two arguments, equivalent to x**y.  With three arguments,
    equivalent to (x**y) % z, but may be more efficient (e.g. for longs).
```

Example:

```
>>> pow(2, 4)
16
>>> pow(2, 4, 3)
1
```

1.3 Functions in expressions and expressions in functions

An **expression** can include not only operators, values, and variables, but also function calls.

```
>>> max(3, 4) * 2 + abs(-3)
8
```

Thus, we can revise our definition of an expression: An **expression** is a combination of operators, values, variables and function calls that python can evaluate, resulting in a single value.

So, functions can appear in expressions. In addition, expressions can appear in the arguments to a function.

```
>>> max(3 ** 4, 4 ** 3)
81
>>> min(3 ** 4, 4 ** 3)
64
```

General principle: anywhere you can put a data value, you can put an expression. This is because python will evaluate the expression to produce a data value. The examples above illustrate this principle: one of the arguments to **`max`** is the expression `3 ** 4` which evaluates to the number 81.