Supplying arguments

When defining a custom function in Python programming you may, optionally, specify an "argument" name between the function's parentheses. A value can then be passed to that argument by specifying the value in the parentheses of the call to the function. The function can now use that passed in value during its execution by referencing it via the argument name. For example, defining a function to accept an argument to print out, like this:

def echo(user) :
**   print('User:' , user)**

A call to this function must specify a value to be passed to the argument within its parentheses so it can be printed out:

echo('Mike')

Multiple arguments (a.k.a. "parameters") can be specified in the function definition by including a comma-separated list of argument names within the function parentheses:

def echo(user , lang , sys) :
**   print('User:' , user , 'Language:' , lang , 'Platform:' , sys)**

When calling a function whose definition specifies arguments, the call must include the same number of data values as arguments. For example, to call this example with multiple arguments:

echo('Mike' , 'Python' , 'Windows')

The passed values must appear in the same order as the arguments list unless the caller also specifies the argument names, like this:

echo(lang = 'Python' , user = 'Mike' , sys = 'Windows')

Optionally, a default value may be specified in the argument list when defining a function. This will be overridden when the caller specifies a value for that argument, but will be used by the function when no value gets passed by the caller:

def echo(user , lang , sys = 'Linux') :
**   print('User:' , user , 'Language:' , lang , 'Platform:' , sys)**

This means you may call the function passing fewer values than the number of arguments specified in the function definition, to use the default argument value, or pass the same number of values as specified arguments to override the default value.