# Functions

A **function** (or sometimes called a ***subroutine***) is a sequence of program instructions that performs a specific task, packaged as a unit. This unit can then be used in programs wherever that particular task should be performed.

## Function Calls

### Calling the `len()` Function

In Python, there is a function named `len`, you can pass in an **argument** into the `len` function like so:

```
# test0.py

L = [1, 3, 4, 5, 6, 7]
print len(L)

# ======
# Output
# ======
6
```

The `len` function allows only a single argument passed into it. The argument can be a *sequence* (i.e. string, bytes, tuple, list, or range) or a *collection* (i.e. dictionary, set, or frozen set). For now, it would be easier to just keep in mind that the `len` function can take in a `list` as its argument.

In the above example, the `len` function's output is the total number of elements in the `L` list. It is common to say that a function **takes** an **argument** and **returns** a result. The *result* is also called the **return value**.

### Calling a Conversion Function (Type Conversion)

Python provides functions that convert values from one type to another.

The `int` function takes any value and converts it to an integer, **if it can**, or complains otherwise:

```
>>> print int('32')
32
>>> print int('Hello')
ValueError: invalid literal for int(): Hello
```

# Function Composition

One of the most useful features of programming languages (not just in Python) is their ability to take small building blocks and **compose** them. For example, an argument of a function can be any kind of **expression**, including the arithmetic operators:

```
# test1.py

import math

z = math.pow(4 / 2, 2) # Returns the 2 raised to the power of 2.

print z

# ======
# Output
# ======
4
```

As you can see from the above example, the first argument to the `pow()` function is the expression: `4 / 2`.

A ***function call is also an expression***, which means that you can call a function within another function as an argument.

```
# test2.py

z = math.pow(math.pow(2, 2), 2)

print z

# ======
# Output
# ======
16
```

# Function Definition

A **function definition** specifies the name of a new function and the sequence of statements that run when the function is called:

```
def abc():
    print "Hello" # print statement ("Hello" is an expression within this statement)
```

`def` is a keyword that indicates that you are about to **def**ine a function. The name of the function is `abc` in the above example, and the rules for a function are the same as the rules for the variable names: letters, numbers, and underscore are legal, but the first character cannot be a number. You cannot use a Python built-in keyword as the name of a function, and you should avoid having a variable and a function with the same name in your own program.

> Function name can have letters, numbers, and underscores, but the first character cannot be a number.

> You should not use a Python keyword as the name for a function.

The empty parentheses after the name of the function `abc` indicate that the function does not take any arguments.

## Function Header & Body

The first line of a **function definition** is called the **header**. In the example above, `def abc():` is the header for the function `abc()`. The indented region that follows the function header is called the **body** of the function. The header needs to end with a colon and the body has to be indented. By convention, indentation is always four spaces. The body can contain any number of statements (any number of lines of code).

# Function Speak

Whenever you are writing or discussing with someone about a function, you can express (in speech or writing) the definition or the usage of a function in the following ways:

- "A function takes an argument and returns a value."
- "A function takes one or more arguments and returns zero or more values."
- "A function has a parameter and it returns a value."
- "A function has one or more parameters and returns one or more values."
- "We can pass in one or more arguments to a function and get back zero or more return values from it."

> When you call a function, you can **pass in zero or more arguments**.

> When you define a function, you **can create parameters** for it. A function **can have zero or more parameters**.