Chapter 5
Functions

5.1 Immediate and Delayed Assignment

This is an ordinary (immediate) assignment. The result of calculation of the right-hand side (in this case, a quadratic polynomial) is assigned to the variable a.

In[1] := a = Expand[(x + 1)^2]

Out[1] = $1 + 2x + x^2$

And this is a delayed assignment. The unevaluated right-hand side (in this case, an expression with the function Expand) is assigned to the variable b.

In[2] := b := Expand[(x + 1)^2]

Note that a delayed assignment returns no value (an ordinary assignment returns the result of calculation of its right-hand side). The difference between a and b can be seen if we assign something to the variable x. In the first case, the value of x is substituted into the quadratic polynomial.

In[3] := x = z + 1; a

Out[3] = $1 + 2(1 + z) + (1 + z)^2$

In the second case, the value of x is substituted into the expression with the function Expand, and then this expression is calculated.

In[4] := b

Out[4] = $4 + 4z + z^2$

In[5] := Clear[a, b, x]

The real name of the operator := is SetDelayed.

In[6] := FullForm[Hold[a := x]]

```
Out[6]//FullForm=
    Hold[SetDelayed[a, x]]
```

Similarly, in addition to ordinary substitutions $a->b$ (where the right-hand side is calculated at the moment the substitution is defined), there are delayed substitutions $a :> b$ (where the substitution keeps the right-hand side unevaluated, and it is calculated each time the substitution is applied).

A. Grozin, *Introduction to Mathematica® for Physicists*, Graduate Texts in Physics,
DOI 10.1007/978-3-319-00894-3_5, © Springer International Publishing Switzerland 2014

In[7] := f[z+1]/.f[x_]->Expand[(x+1)^2]
Out[7] = $1+2(1+z)+(1+z)^2$
In[8] := f[z+1]/.f[x_]:>Expand[(x+1)^2]
Out[8] = $4+4z+z^2$

5.2 Functions

Left-hand side of an assignment can be a pattern, not just a variable. In this case, in all subsequent calculations, any subexpression matching the pattern will be replaced by the right-hand side of the assignment. This can be canceled by the command Clear. A pattern can contain arbitrary variables. This is how functions are defined. Here is an example—a function f. In all subsequent calculations, all subexpressions of the form $f[x]$ with arbitrary arguments x will be replaced by the right-hand side (in this case, a quadratic polynomial), in which the value of the actual argument is substituted for x.
In[9] := f[x_] = Expand[(x+1)^2]
Out[9] = $1+2x+x^2$
And this is another function. Its body is an unevaluated expression with Expand. Expanding brackets will take place each time the function g with some argument is calculated.
In[10] := g[x_] := Expand[(x+1)^2]
Note the difference between them.
In[11] := {f[z+1],g[z+1]}
Out[11] = $\{1+2(1+z)+(1+z)^2, 4+4z+z^2\}$
In[12] := Clear[f,g]

5.3 Functions Remembering Their Values

Let's consider a useful trick—a function remembering its calculated values. If it is called again with the same argument, it will not perform calculations, but just return the remembered result. For example, take the factorial. We know fac[0].
In[13] := fac[0] = 1
Out[13] = 1
And now attention—the main trick. A delayed assignment to the pattern fac[n_] (with an arbitrary n). And what do we have in the right-hand side? An immediate assignment to the pattern fac[n] (for a specific n, namely, the value of the actual argument with which the function fac was called). What happens when we call fac[10]? If the function was never calculated with this argument, then this definition for an arbitrary argument will be used. The right-hand side with 10 substituted for n will be calculated, namely, an immediate assignment fac[10] = ⋯. Its right-hand side is calculated (it is the factorial of 10), and it is remembered as the value of fac[10].

The immediate assignment returns the calculated value of its right-hand side, and this value becomes the result of the function call. If we ask *Mathematica* to calculate fac[10] again, then this specific definition for fac[10] (generated during the first calculation) will be used, and not the general definition for fac[$n\_$].

In[14] := fac[$n\_$] := fac[n] = n * fac[$n-1$]

What does *Mathematica* know about the symbol fac?

In[15] :=?fac

Global`fac

fac[0] = 1

fac[$n\_$] := fac[n] = n fac[$n-1$]

Only two definitions—for the argument 0 and for an arbitrary argument. Now let's calculate the factorial of 10.

In[16] := fac[10]

Out[16] = 3628800

And what does *Mathematica* know about this symbol now?

In[17] :=?fac

Global`fac

fac[0] = 1

fac[1] = 1

fac[2] = 2

fac[3] = 6

fac[4] = 24

fac[5] = 120

fac[6] = 720

fac[7] = 5040

fac[8] = 40320

fac[9] = 362880

fac[10] = 3628800

fac[$n\_$] := fac[n] = n fac[$n-1$]

In addition to the general definition, we see also specific ones for all integer values of the argument from 0 to 10. If we ask for the value of fac for one of these arguments, then the corresponding specific definition will be used, and the calculation will not be performed again.

In[18] := Clear[fac]

5.4 Fibonacci Numbers

This method is useful but not vital for the factorial, because the time of calculation of fac[n] grows linearly with n. For Fibonacci numbers the difference is crucial. For a naive definition, the calculation time grows exponentially. This means you will never get a Fibonacci number with a large n. When results are remembered, the calculation time grows linearly—the result for each value of the argument from 2 to n is calculated once.

In[19] := fib[0] = fib[1] = 1
Out[19] = 1
In[20] := fib[n_] := fib[n] = fib[n − 1] + fib[n − 2]
In[21] :=?fib
Global`fib
fib[0] = 1
fib[1] = 1
fib[n_] := fib[n] = fib[n − 1] + fib[n − 2]
In[22] := fib[10]
Out[22] = 89
In[23] :=?fib
Global`fib
fib[0] = 1
fib[1] = 1
fib[2] = 2
fib[3] = 3
fib[4] = 5
fib[5] = 8
fib[6] = 13
fib[7] = 21
fib[8] = 34
fib[9] = 55
fib[10] = 89
fib[n_] := fib[n] = fib[n − 1] + fib[n − 2]
In[24] := Clear[fib]

5.5 Functions from Expressions

In most cases, a delayed assignment is used when defining a function. But there are situations when an immediate assignment is needed. Here is one of them. Suppose you have derived an expression a containing a symbol x as a result of some calculation.

In[25] := a = D[Expand[(x + 1)^3],x]
Out[25] = $3 + 6x + 3x^2$

Now you want to calculate it many times with different values of x. This can be done by substitutions.

In[26] := a/.x−>z + 1
Out[26] = $3 + 6(1 + z) + 3(1 + z)^2$

But this is not very convenient. It would be nice to have a function f with the argument x which is given by the calculated expression a. Such a function can be defined by an immediate assignment. The calculated value is substituted for a in the right-hand side.

In[27] := f[x_] = a
Out[27] = $3+6x+3x^2$
In[28] := f[z+1]
Out[28] = $3+6(1+z)+3(1+z)^2$
In[29] := Clear[a, f]

5.6 Antisymmetric Functions

It is often useful to give a partial definition of a function. To this end we write not just a function with all arguments being arbitrary but a more restrictive pattern in the left-hand side of an assignment. Then, if the values of the actual arguments match the pattern, the function is calculated (i.e., replaced by the right-hand side of the assignment). Otherwise the function remains unevaluated. Here is a simple example. Let's define an antisymmetric function of two arguments. If the arguments are equal, it vanishes.
In[30] := f[x_, x_] := 0
If they are not equal, we have to decide if they need to be interchanged. The expressions $f[x,y]$ and $f[y,x]$ should reduce to either the first form or the second one, for any x and y. It does not matter to which form, as long as the result is always the same. To this end the function OrderedQ is useful. Its argument is a list. It returns True if the list is ordered, i.e., each element is "greater than or equal to" the previous one in the sense of some internal ordering *Mathematica* uses for expressions. Details of this ordering are not important.
In[31] := {OrderedQ[{x, y}], OrderedQ[{y, x}], OrderedQ[{x, x}]}
Out[31] = {True, False, True}
Now it is easy to write a substitution which interchanges the arguments if they are not properly ordered.
In[32] := f[x_, y_] /; Not[OrderedQ[{x, y}]] := -f[y, x]
In[33] := {f[a, a], f[a, b], f[b, a]}
Out[33] = $\{0, f[a,b], -f[a,b]\}$
In[34] := {f[a + b, a - b], f[a - b, a + b]}
Out[34] = $\{-f[a-b,a+b], f[a-b,a+b]\}$
In[35] := Clear[f]

Of course, a symmetric function can be defined similarly. An odd function of a single argument can be defined in the same way.
In[36] := f[0] = 0
Out[36] = 0
In[37] := f[x_] /; Not[OrderedQ[{-x, x}]] := -f[-x]
In[38] := {f[0], f[a], f[-a]}
Out[38] = $\{0, f[a], -f[a]\}$
In[39] := {f[a - b], f[b - a]}
Out[39] = $\{-f[-a+b], f[-a+b]\}$
In[40] := Clear[f]
Of course, an even function can be defined similarly.

5.7 Functions with Options

You have undoubtedly noted that many *Mathematica* functions (e.g., Plot) have options. They can be specified in any order; each option is given by a substitution with its name in the left-hand side and its value in the right-hand side. If they are not given, their default values are used. Suppose you want your own function f to have options. This can be done in the following way. Let's assign a list of substitutions giving default values of all options to Options$[f]$. Define the function f with some mandatory arguments and an arbitrary sequence of arguments opts___ (it may be empty). At the point in the function body where you need the value of the option opt1 use opt1/.{opts}/.Options$[f]$. The operations /. are executed left to right. Therefore, if the user has included a substitution opt1 $\to \cdots$ among the arguments, the left /. will trigger, and the result will be some value which contains no option names; the right /. will not change it. If the user has not given such a substitution, the left /. will do nothing, and the right one will replace opt1 by the default value of this option.

In[41] := Options[f] = {opt1->1, opt2->2}
Out[41] = {opt1 → 1, opt2 → 2}
In[42] := f[x_, opts___] := g[x, opt1/.{opts}/.Options[f],
opt2/.{opts}/.Options[f]]
In[43] := {f[a], f[a, opt2->0], f[a, opt2->b, opt1->c]}
Out[43] = {g[a, 1, 2], g[a, 1, 0], g[a, c, b]}
In[44] := Clear[f]

In recent versions of *Mathematica* this can also be written as follows:

In[45] := f[x_, OptionsPattern[f]] := g[x, OptionValue[opt1], OptionValue[opt2]]
In[46] := Options[f] = {opt1->1, opt2->2};
In[47] := {f[a], f[a, opt2->0], f[a, opt2->b, opt1->c]}
Out[47] = {g[a, 1, 2], g[a, 1, 0], g[a, c, b]}
In[48] := Clear[f]

5.8 Attributes

A function can have attributes which affect simplification of expressions with this function. The attribute Flat removes nested function calls (e.g., Plus and Times have this attribute).

In[49] := Attributes[f] = {Flat}
Out[49] = {Flat}
In[50] := f[x, f[y, z], u]
Out[50] = f[x, y, z, u]

The attribute Orderless means that the function is symmetric in all arguments, and *Mathematica* may interchange them at will (Plus and Times have also this attribute).

In[51] := Attributes[f] = {Orderless}
Out[51] = {Orderless}
In[52] := {f[x,y,z], f[z,x,y], f[y,z,x]}
Out[52] = {f[x, y, z], f[x, y, z], f[x, y, z]}
The attribute Listable means that if the first argument is a list, then the function is applied to each element, and the list of results is returned (Plus and Times have this attribute, too).
In[53] := Attributes[f] = {Listable}
Out[53] = {Listable}
In[54] := f[{x,y,z}]
Out[54] = {f[x], f[y], f[z]}
In[55] := f[{x,y,z}, a]
Out[55] = {f[x, a], f[y, a], f[z, a]}
There exist a few attributes more. Of course, a function can have several attributes at once. The command Clear[f] removes only substitutions for f (with any arguments), but not its attributes. In order to remove attributes too, use ClearAll.
In[56] := ClearAll[f]; Attributes[f]
Out[56] = {}

5.9 Upvalues

Suppose we want to define a function f such that $f[x] * f[y]$ is replaced by $f[x+y]$ for arbitrary x and y. This can be done by the assignment $f[x\_] * f[y\_] := f[x+y]$. This definition will be associated with the function Times; *Mathematica* will have to check it each time it multiplies something, i.e., very often, and performance will degrade. It is possible to associate this definition with the function f instead. Then it will be used only when processing a product containing at least one function f.
In[57] := f[x_] * f[y_]^ := f[Expand[x + y]]
In[58] := f[(x+y)^2] * f[(x-y)^2] * f[x^2] * g[y^2]
Out[58] = $f\left[3x^2 + 2y^2\right] g\left[y^2\right]$
Here the left-hand side is Times[f[x_], f[y_]], and the definition is associated with f. If we want a definition for Times[f[x_], g[y_]], we can associate it with either f or g.
In[59] := f/: f[x_] * g[y_] := f[Expand[x - y]]
In[60] := f[(x+y)^2] * f[(x-y)^2] * f[x^2] * g[y^2]
Out[60] = $f\left[3x^2 + y^2\right]$
In[61] :=?f
Global`f
f[x_]f[y_]^ := f[Expand[x + y]]
f/: f[x_]g[y_] := f[Expand[x - y]]

When processing an expression f[g1[...], g2[...], g3[...]], *Mathematica* uses definitions associated with f and also definitions associated with g1, g2, g3, and

having the form $f[\ldots] := \cdots$ (*upvalues* of g1, g2, g3). It does not look deeper, into arguments of g1, g2, and g3—this would be too inefficient. In addition to the delayed assignments $^\wedge :=$ and $f/:$ lhs $:=$ rhs there are also immediate assignments $^\wedge =$ and $f/:$ lhs $=$ rhs.

In[62] := Clear[*f*]