

4. Writing Functions



4.1 Introduction

Objectives

4.2 Understanding Function Definitions

The need for patterns

The need for delayed assignment

4.3 Clearing Function Definitions

Why it is important to clear old function definitions

How to ensure you don't go crazy

4.4 An Example with Three Variables

Defining the load capacity of a thrust bearing

Using the formula for the load capacity

4.5 Summary: The Form of a Function Definition

Summary

Exercises

4.6 Problems

Problem 1: Computing amplitudes of vibration

Problem 2: Reflecting graphs

Problem 3: Rotating graphs

Problem 4: Checking algorithms

Problem 5: Calculating data averages

4.1 Introduction

In this section on writing function definitions, we begin programming in full strength. When you write a function definition, you have written a program. It may be a short one, but sometimes you can write one-line programs in Mathematica which would take hundreds of lines of code in another language. There is nothing wrong with a short program!

Being able to write a good function definition is *absolutely critical* to your ability to program with Mathematica. Make sure you understand this module completely!

Objectives

- To understand why functions are defined as they are.
- To be able to write function definitions.
- To demonstrate good practice in the inclusion of **Clear** command with each function definition.

4.2 Understanding Function Definitions

The need for patterns

The expression $\mathbf{x^3 + 2 x^2 - 3}$ is mathematically a *function* of \mathbf{x} . We might try defining a "shorthand" for this expression in the form $\mathbf{f[x]}$ by entering

$$\mathbf{f[x] = x^3 + 2 x^2 - 3}$$

$$- 3 + 2 x^2 + x^3$$

If we enter $\mathbf{f[x]}$ we get back $\mathbf{x^3 + 2 x^2 - 3}$ as we expect (remember, do not worry about the different ordering of the terms, it makes no difference)

$$\mathbf{f[x]}$$

$$- 3 + 2 x^2 + x^3$$

But this definition only works for a function argument of \mathbf{x} and no other, for example, $\mathbf{2}$ or \mathbf{a} .

$$\{\mathbf{f[x], f[2], f[a]}\}$$

$$\{- 3 + 2 x^2 + x^3, f[2], f[a]\}$$

If $\mathbf{f[x]}$ is to behave like a function, we would expect it to return the *right hand side* with the correct value of the argument substituted. We can do this by using a *pattern* $\mathbf{x_}$ instead of \mathbf{x} for the argument

$$\mathbf{f[x_] = x^3 + 2 x^2 - 3}$$

$$- 3 + 2 x^2 + x^3$$

This now means that there is nothing special about \mathbf{x} in particular, that it is just acting as a placeholder for any expression. We now get the behaviour we were expecting for different arguments

$$\{\mathbf{f[x], f[2], f[a], f[x+2]}\}$$

$$\{- 3 + 2 x^2 + x^3, 13, - 3 + 2 a^2 + a^3, - 3 + 2 (2 + x)^2 + (2 + x)^3\}$$

The need for delayed assignment

So far, so good. But consider the following function definition

```
f[x_] = Expand[x2]  
x2  
  
{f[x], f[2], f[a], f[x + 2]}  
  
{x2, 4, a2, (2 + x)2}
```

The argument **x+2** has not been expanded. Yet that is just what we wanted the function to do. What has happened is that because we used **Set** (=) instead of **SetDelayed** (:=), the right hand side was evaluated *at the time we entered the definition* to give, simply **x**² on the right hand side. Thus, our definition was really equivalent to **f**[**x_**] = **x**².

If now, we define our function with :=, the Expand will not be applied to the square of the argument until the actual argument is entered.

```
f[x_] := Expand[x2]  
  
{f[x], f[2], f[a], f[x + 2]}  
  
{x2, 4, a2, 4 + 4 x + x2}
```

This is the behaviour we want.

4.3 Clearing Function Definitions

Why it is important to clear old function definitions

One of the powerful programming methods built into the Mathematica language is that of rule-based programming (which we look at in the next module). This means that it remembers every new rule (function definition) that you enter for a function symbol (like **f** in the examples above) rather than overwriting the previous ones. We will see that using multiple rules (function definitions) can be a very powerful way to program, if you define your rule set with this capability in mind. In the meantime, however, we are looking at *just one function definition* for each function symbol (like **f**), and if we have entered more than one, it is because all but one were incorrect, and not what we wanted.

It is *imperative* therefore that as you design and develop a function definition (try something, find one aspect does not work, modify it, ...), *you clear out the previous incorrect definition* from the Kernel's memory before defining the new one. You clear the definition associated with the function symbol **f** by entering **Clear**[**f**].

Keep your function development on track by using **Clear**[□]

How to ensure you don't go crazy

A good habit *which we highly recommend* as a way of staying sane in the face of frustration caused by *forgetting* to **Clear** a definition is to begin defining every function by putting the command **Clear[]** in the same cell, before your definition. Separate them by a blank line and enclose them in a box for clarity. (To get a box around the contents of a cell, select the cell bracket, and then the menu item Format:Background Color:Cell Gray Box. Then select Background Color:None). For example, our previous function definition should look like

```
Clear[f]

f[x_] := Expand[x^2]
```

Now, every time you press Shift-Enter with the cursor *anywhere* in the cell, **Clear[f]** is entered first and thus your new definition is entered afresh (and you stay sane!).

4.4 An Example with Three Variables

Defining the load capacity of a thrust bearing

Suppose you were interested in the load capacity of a circular thrust bearing, the formula for which is

$$\frac{\pi P (R^2 - r^2)}{2 \text{Log} \left[\frac{R}{r} \right]}$$

Here, **P** is the lubricant pressure, **R** is the outer radius, and **r** is the inner radius.

The load capacity is clearly a *function* of the three variables, **P**, **R**, and **r**. We can define such a function (say **W**) by entering

```
Clear[W]

W[P_, R_, r_] :=  $\frac{\pi P (R^2 - r^2)}{2 \text{Log} \left[ \frac{R}{r} \right]}$ 
```

Using the formula for the load capacity

Once such an expression has been entered, the (simpler) expression on the left hand side can then be used instead of the more complicated one on the right hand side. For example, we can retrieve the original formula by entering **W[P, R, r]**

W[P, R, r]

$$\frac{P \pi (-r^2 + R^2)}{2 \text{Log} \left[\frac{R}{r} \right]}$$

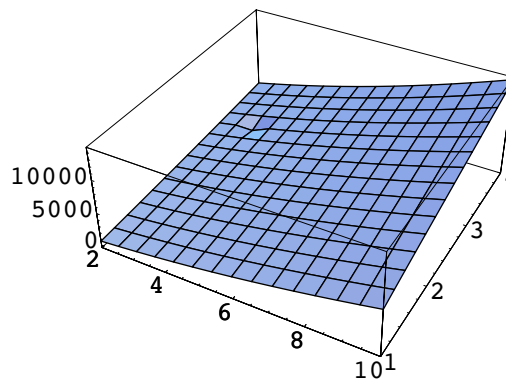
Or, we can evaluate the load capacity for specific values of its variables

W[100, 10, 2.5]

10622.7

Or, we can plot it as a function of the inner and outer radii for a fixed pressure

Plot3D[W[100, R, r], {R, 2, 10}, {r, 1, 4}]



- SurfaceGraphics -

4.5 Summary: The Form of a Function Definition

Summary

In sum, the form of a function definition is then

```
Clear[f]
```

```
f[x_, y_, z_, ...] := some expression involving x, y, z, ...
```

- The symbol **f** is the *name* of the function and **x, y, z, ...** are its *arguments*.
- It is good practice to have *all* the different symbols which appear on the right hand side as arguments on the left hand side.
- The *actual* symbols chosen make no difference. For example, we could choose α, β, γ , instead of **x, y, z**.

In fact, even if you have previously defined **x = 2**, it will not conflict with the function

definition, since Mathematica recognizes that the symbols used in function definitions are just *placeholders* for *anything* you might care to insert in that place.

Any expression that you develop can be redefined as a function simply by writing a left hand side to it of the form $\mathbf{f}[\mathbf{x}_-, \mathbf{y}_-, \mathbf{z}_-, \dots] :=$

Exercises

◆ Exercise: Defining a function of one variable

Define a function $\mathbf{h}[\mathbf{z}]$ which takes the list of functions below and evaluates them.

$$\{\mathbf{Log}[\mathbf{z}], \mathbf{Sin}[\mathbf{z}], \mathbf{z Tan}[\mathbf{z}], \mathbf{z}^2 \mathbf{Exp}[\mathbf{z}]\}$$

◆ Exercise: Defining a function of two variables

Define a function $\mathbf{d3}[\mathbf{y}, \mathbf{x}]$ which lists the first three derivatives of an expression \mathbf{y} . (Assume \mathbf{y} is a function of \mathbf{x}).

◆ Exercise: Defining a function of several variables

Define a function \mathbf{T} for the torque on a clutch. The torque is given by the formula

$$\frac{2}{3} \mu \mathbf{F} \left(\frac{\mathbf{R}^3 - \mathbf{r}^3}{\mathbf{R}^2 - \mathbf{r}^2} \right) \mathbf{n}$$

Plot the function for fixed values of μ , \mathbf{F} , and \mathbf{n} using **Plot3D**.

◆

4.6 Problems

Problem 1: Computing amplitudes of vibration

The amplitude of a damped vibration in a suspension system is given by the formula $\mathbf{8Sin}[\pi\mathbf{t}]\mathbf{Exp}[-\mathbf{t}]$.

Define a function **amplitude** $[\mathbf{t}]$ for the amplitude as a function of \mathbf{t} .

Test out your function definition by plotting both the original formula $\mathbf{8Sin}[\pi\mathbf{t}]\mathbf{Exp}[-\mathbf{t}]$, and **amplitude** $[\mathbf{t}]$ with \mathbf{t} from $\mathbf{0}$ to $\mathbf{4}$.

Use the inbuilt function **SameQ** (find out about it by entering **?SameQ**) to see if these plots (that is, the points in their **FullForm** expressions) are *exactly* the same.



Problem 2: Reflecting graphs

Define a function **reflect** for reflecting a graph in the line $y = x$.

Hint: Do not forget to include **Show** in your function definition.

Test out your **reflect** function by using the **amplitude** function developed in the previous problem.

Test using **SameQ** to see if the application of your **reflect** function twice returns you to the original amplitude plot.



Problem 3: Rotating graphs

Define a function **rotate** for rotating a graph *counterclockwise* by 90° .

Test out your **rotate** function by using the **amplitude** function developed in the previous problem.

Test using **SameQ** to see if the application of your **rotate** function *four* times returns you to the original amplitude plot.



Problem 4: Checking algorithms

Write a one-line function which checks *Mathematica's* ability to integrate correctly the derivative of a given function of one variable (with possible symbolic constants), returning either **True** or **False**.

Test your function thoroughly.



Problem 5: Calculating data averages

Write a one-line function that calculates the average of the elements of a list of any length.

Test your function thoroughly.

Hints: Use your understanding of **FullForm** to see how you can turn a list into a sum of its elements. Look at the inbuilt function **Length**.

