8 Chapter 8 Programming with functions

8.1 Chapter overview

Most computer languages regard functions or procedures as the "lego blocks" for building programs. Not only is it expected that there will be a lot of different kinds of blocks that will be provided, but that you will build things by putting them several of them together. The way functions are used in this way is through daisy chaining -- by making the output of one function the input of another. These chains can then be defined to be functions themselves. By defining a few chains and then using them together, powerful combinations of operations can be custom-built quickly for the user's needs.

Most computer languages extend the concept of functions to go beyond the numbers or formulas that "mathematical functions" provide. In Maple, as in most of languages, a function can return other kinds of results. It is fairly common in Maple and other languages to have functions that produce as output a list, an equation, or a string as a result. As we shall see, we can even return a Maple plot as the result of a function. In a symmetric fashion, it is possible for computer functions to have lists, equations, plots, or strings as inputs.

8.2 Designing functions from context

In doing technical work, we often see functions defined as an equation relating the name of the function, its argument(s), and the function definition. Those are easy to translate into Maple's notation and use. For example if we see in a mathematics book "define \( f(x) = x^2 + 2x - v0 \)" then we can just transcribe it into the Maple function notation: \( f := (x) \rightarrow x^2 + 2x - v0 \).

In word problems, we have to "read between the lines" and design the function. This requires answering the questions:

a) What will the inputs to the function be? Try to give symbolic names for it.

b) What will the output be? Sometimes to realize what the output is, you can create a worksheet with several steps. If there is only one final result, then that should be the output.

c) How do you calculate the output from the inputs? Hopefully, there's a simple formula that describes this.

We illustrate this process with an example:

**Designing a function for pressure/temperature problems**


The Ideal Gas Law, as stated in *Introduction to Engineering* is:

\[
P \cdot V = n \cdot R \cdot T
\]

\[
P V = n R T
\]  

(8.1)

where

\( P \) is pressure in Pascals (Pa)

\( V \) is volume in \( m^3 \).

\( n \) is the amount of gas in moles (mol),

\( T \) is the temperature in degrees K,

\( R \) is the gas constant, approximately \( 8.31 \cdot \frac{m^3 \cdot Pa}{K \cdot mol} \).
We want to solve the following problem (actually, various versions of it):

**Problem**

We measure the temperature and pressure of a gas. It has a pressure of 100 \( \text{[kPa]} \) and a temperature 473 degree Kelvin. Then cool it so it has a pressure of 89.8 \( \text{[kPa]} \). What is its temperature?

![Diagram of atmospheric pressure and ideal gas law](source: Introduction to Engineering by Jay Brockman, Wiley, 2009 (p. 180))

**Finding the answer**

First, we do the mathematical thinking and informal calculation that allows us to build a function that will solve all problems of this type:

For a fixed cylinder volume, according to the Ideal Gas Law:

\[
\frac{P_1}{T_1} = \frac{P_2}{T_2}, \text{ so } \]

\[
T_2 = \frac{P_2}{P_1} \cdot T_1. \text{ so } \]

\[
T_2 = \frac{89.8}{100} \cdot 473 \]

\[
= 424.754000 \]

**Function design**

We see that the problem wants us to calculate a temperature \( T_2 \) given the atmospheric pressure \( P_1 \), the internal pressure \( P_2 \), and the first temperature, \( T_1 \).

The output is \( T_2 \), the inputs are \( P_1 \), \( P_2 \), and \( T_1 \). We are free to name the function anything we want since the problem statement doesn't name this. We decide to call it something that reminds us of the purpose.

\[
\text{secondTemp} := (P_1, P_2, T_1) \rightarrow \frac{P_2}{P_1} \cdot T_1
\]
(P1, P2, T1) → \( \frac{P2T1}{P1} \) \hspace{1cm} (8.2)

**Testing and troubleshooting the function**

We see whether we get the intended result with the numbers we've already worked out. Note that if we hadn't done the analysis, we wouldn't have any way of testing what we designed.

\[ \text{secondTemp}(89.8, 100, 473) \]

\[ 526.7260579 \] \hspace{1cm} (8.3)

Oops, that isn't the same result. What did we do wrong? The formula 1.2.2 seems like the right thing. What else could go wrong? Close inspection indicates that the first argument to internalTemp is P1, which appears in the denominator of the formula for the output. In (1.2.3), that would put the "89.8" in the denominator, but our example had 89.8 in the numerator. Oops, we gave the values in the wrong order for the function. There's nothing wrong except that we should invoke the function with the information given in the correct order:

\[ \text{secondTemp}(100, 89.8, 473) \]

\[ 424.7540000 \] \hspace{1cm} (8.4)

**Using the function**

We are given a different version of the problem:

We measure the temperature and pressure of a gas. It has a pressure of 2000 \( [kPa] \) and a temperature 473 degree Kelvin. Then cool it so it has a pressure of 53.6 \( [kPa] \). What is its temperature?.

Answer:

\[ \text{secondTemp}(2000, 56.6, 473) \]

\[ 13.38590000 \] \hspace{1cm} (8.5)

Since the answer is in degrees Kelvin, this is only about 14 degrees above absolute zero. That's pretty cold!

**The usefulness of alternative function designs**

Suppose we had this new problem:

We measure the temperature and pressure of a gas. It has a pressure of 100 \( [kPa] \) and a temperature 473 degree \( [K] \). We then heat it to 512 degrees Kelvin. What is its pressure then?

**Another function designed**

A little thought produces the calculation:

\[ P2 = \frac{P1}{\frac{T1}{T2}} = \]

\[ \frac{100}{\frac{473}{512.0}} \]
This leads to the function definition:

\[ \text{secondPressure} := (P1, T1, T2) \rightarrow \frac{P1}{T1} \frac{T2}{T2} \]

\[ (P1, T1, T2) \rightarrow \frac{P1 T2}{T1} \]  

(8.6)

We test this (remembering what happened before about the order of arguments)

\[ \text{secondPressure}(100, 473, 512.0) \]

\[ 108.2452431 \]  

(8.7)

**Conclusion**

To develop functions, it helps to have worked through some typical calculations interactively. Once you have realized which quantities you are starting with and named them, and have developed the formula for the calculation using those names, you can create a function definition. You can use the names given in the problem description, or you can make up names based on their purpose. Unlike mathematics, you are not limited to single letters for names of variables or names of functions. Computer programmers know that longer names are often easier to remember or understand.

### 8.3 Function composition: daisy-chaining functions together

In the scripts we have developed so far, we have developed a result through a sequences of actions. These sequences can often be described through functional *composition* — an expression that chains together several actions. Consider the following example:

**Problem**

On November 1, 2007, one Euro was worth 1.002908434 US dollars. We are buying widgets that cost 30 Euros each and importing them into the US. We then put the widgets into packages that cost .075 dollars. How much should we sell the product in the US if we want a 10% markup from our costs?

**Finding the solution: step 1, first do the calculation interactively.**

Doing this in the style of scripts, we first assign the values to variables, and then do the calculational steps.

\[ \text{convRate} := 1.002908434 \]

\[ 1.002908434 \]  

(8.8)

\[ \text{costInEuros} := 30 \]

\[ 30 \]  

(8.9)

\[ \text{pkgCost} := .075 \]

\[ 0.075 \]  

(8.10)

\[ \text{markupPct} := .10 \]
0.10

\( totalCost := pkgCost + \text{convRate} \cdot \text{costInEuros} \)

30.16225302

\( sellingPrice := (1 + \text{markupPct}) \cdot totalCost \)

33.17847832

We foresee using this calculation several times as the conversion Rate, the manufacturing cost in Europe, and the packaging cost change. We even see that the markup might change. We can try to boil down these steps into a few functions.

**Designing the solution: step 2, design functions to do the calculational steps**

\( totalCostFunc := (\text{convRate}, \text{costInEuros}, \text{pkgCost}) \rightarrow (\text{pkgCost} + \text{convRate} \cdot \text{costInEuros}) \)

\( priceFunc := (\text{markupPct}, \text{totalCost}) \rightarrow (1 + \text{markupPct}) \cdot \text{totalCost} \)

\( sellingPriceFunc := (\text{convRate}, \text{costInEuros}, \text{pkgCost}, \text{markupPct}) \rightarrow priceFunc(\text{markupPct}, totalCostFunc(\text{convRate}, \text{costInEuros}, \text{pkgCost})) \)

Note that the way that the third function \( sellingPriceFunc \) is defined, it takes the output of \( totalCostFunc \) and makes it one of the inputs to \( priceFunc \).

**Testing the solution: step 3, test the building blocks in the order that they are used**

We test the first two functions. After we see that they agree with our preliminary version of the calculations, we test the third function that depends on the correctness of the first two.

\( totalCostFunc(\text{convRate}, \text{costInEuros}, \text{pkgCost}) \)

30.16225302

\( priceFunc(0.10, (1.3.10)) \)

33.17847832

\( sellingPriceFunc(\text{convRate}, \text{costInEuros}, \text{pkgCost}, \text{markupPct}) \)

33.17847832

We could write a script that just used \( totalCostFunc \) and \( priceFunc \), but by designing and using a third function, we reduce the work of handling an instance of the problem to just pasting in the values for the four parameters in one line. This is probably less work than changing four lines of parameters that we have had with our previous approach to scripts.
While function composition is a succinct way of ordering many operations, its advantages are apparent only after the chain is built and tested as working correctly. It maybe easier to develop the chain as a script of assignments and then refactor -- rewrite without changing the meaning -- the script so that it uses user-defined functions to replace some of the chains of assignments.

Using the solution: step 4, present a script that defines the functions, then invoke the "answer function" repeatedly to handle various versions of the problem.

A script that uses functional composition (chaining), and its use

Begin function definitions

\[\text{totalCostFunc} := (\text{convRate}, \text{costInEuros}, \text{pkgCost}) \rightarrow (\text{pkgCost} + \text{convRate} \cdot \text{costInEuros})\]
\[\text{priceFunc} := (\text{markupPct}, \text{totalCost}) \rightarrow (1 + \text{markupPct}) \cdot \text{totalCost}\]
\[\text{sellingPriceFunc} := (\text{convRate}, \text{costInEuros}, \text{pkgCost}, \text{markupPct}) \rightarrow \text{priceFunc}(\text{markupPct}, \text{totalCostFunc}(\text{convRate}, \text{costInEuros}, \text{pkgCost}))\]

End function definitions

Problem solving

Version 1

On November 1, 2002, one Euro was worth

\[
\frac{1}{.9971} = 1.002908434
\]

US dollars. We are buying widgets that cost 30 Euros each and importing them into the US. We then put the widgets into packages that cost .075 dollars. How much should we sell the product in the US if we want a 10% markup from our costs?

\[\text{sellingPriceFunc}(1.002908434, 30, .075, .10)\]
\[\text{\$33.18}\]

(We got the number formatted to currency by right-click->Numerical Formatting->Currency.)

Version 2

On November 1, 2007, one Euro was worth 1.4487 US dollars. We are buying widgets that cost 33 Euros each and importing them into the US. We then put the widgets into packages that cost .09 dollars. How much should we sell the product in the US if we want a 10% markup from our costs?

\[\text{sellingPriceFunc}(1.4467, 33, .09, .10)\]
\[\text{\$52.61}\]

Version 3
A script that uses functional composition (chaining), and its use

On November 1, 2009, one Euro was worth 1.4728 US dollars. We are buying widgets that cost 35 Euros each and importing them into the US. We then put the widgets into packages that cost .10 dollars. How much should we sell the product in the US if we want a 10% markup from our costs?

\[ \text{sellingPriceFunc}(1.4728, 35, .10) \]

\[ \text{\textdollar}56.81 \]  (8.25)

We see that if we are interested in looking at the solution to several different versions of the problem, setting up the script as a collection of function definitions presents the problem-solving calculation only once. We can then proceed and present the several solutions through a single-line calculation. We don't have to wade through all the steps of the calculation to see the answer to the first problem, then looking through the same steps to see the answer to the second, etc.

### 8.4 Expressions with units of measurements: convert

Maple has facilities for converting between various English and metric units. It is useful for doing multi-step calculations because the conversions happen automatically.

In the first way of using `convert`, one thinks of a value as implicitly expressing a number of units and wants another number expressing those number of units converted to another unit. One uses `convert(value, units, fromUnit, toUnit).

<table>
<thead>
<tr>
<th>Examples of Unit Conversion</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Examples of unit conversion</strong></td>
</tr>
<tr>
<td>How many inches in a meter? How many feet in a kilometer? How many millimeters in a mile?</td>
</tr>
<tr>
<td>[ \text{convert}(1, \text{units, inch, meter}) ]</td>
</tr>
<tr>
<td>[ \text{convert}(1, \text{units, ft, km}) ]</td>
</tr>
<tr>
<td>[ \text{convert}(1.0, \text{units, mile, mm}) ]</td>
</tr>
<tr>
<td>[ \text{convert}(5.4, \text{units, kilowatt, horsepower}) ]</td>
</tr>
<tr>
<td>[ \text{convert}(2.0, \text{units, angstroms, micrometers}) ]</td>
</tr>
<tr>
<td>[ \text{convert}(15.0, \text{units, miles, meters}) ]</td>
</tr>
</tbody>
</table>

Maple can convert between most compatible units. Sometimes units are expressed as ratios of other units. Maple can handle such conversions as well.
### Examples of unit conversion

$$\text{convert} \left(13.3, \text{units, \frac{\text{gallons}}{\text{yards}^3}, \frac{\text{liters}}{\text{meter}^3}}\right)$$

$$65.85005144$$ (8.32)

In many examples in the Maple documentation, some of the arguments to convert are quoted -- surrounded by apostrophes -- to prevent evaluation from using the value of the names of the units. For example, if you have assigned a value to the variable \(s\), then you cannot convert to seconds with this name without quotation.

#### Troubleshooting unit conversion

\[
\text{seconds} := \text{convert}(3, \text{units, days, seconds})
\]

\[259200\] (8.33)

As long as the various names used as arguments to the \text{convert} function don't have values, things work fine.

\[
\text{seconds2} := \text{convert}(4, \text{units, minutes, seconds})
\]

Error, (in convert/units) unable to convert 'min' to '259200'

Maple performs evaluation of names as it figures out what the inputs to convert is. Since \text{seconds} has a value, Maple tries to compute \text{convert}(4, \text{units, minutes, 259200}). Since the 4th argument to \text{convert} has to be a name, an error results.

\[
\text{seconds2} := \text{convert}(4, \text{units, minutes,'seconds'})
\]

\[240\] (8.34)

Quoting the 4th argument causes the name \text{seconds} to be given as the 4th input to \text{convert}. This works.

\[
\text{seconds3} := \text{convert}(5, \text{'units', 'hours', 'seconds'})
\]

\[18000\] (8.35)

Quoting all the names as a prophylactic measure is acceptable. You see this in a lot of the Maple on-line documentation.

In *Star Wars Episode IV: A New Hope*, Han Solo says that the Millennium Falcon made the Kessel Run in "less than twelve parsecs". We want to know how many days a parsec is.

\[
\text{convert}(12.0, \text{units, parsecs, days})
\]

Error, (in convert/units) unable to convert 'pc' to 'd'

This is the error message you see when you are trying to convert between incompatible units, e.g. trying to convert a gallon into a meter. \text{pc} seems to be Maple's internal name for parsec, \text{d} the name for days.

\[
\text{convert}(12.0, \text{units, parsecs, miles})
\]

\[2.300821388 \times 10^{14}\] (8.36)

A parsec is a non-fictional unit of distance, not time, so we can convert 12 parsecs to miles, kilometers, inches... But we can't convert it to days any more than we can convert inches to volts.

### A problem solved, a script built using function definitions

A car travels a 45 miles per hour. How many minutes does it take to travel 900 kilometers?

\[
distance := 900.0
\]

\[900.0\] (8.37)

\[
speed := 45
\]

\[45\] (8.38)

\[
d := \text{convert}(\text{distance, units, kilometers, miles})
\]

\[559.2340730\] (8.39)

We build a sequence of calculations to understand how to solve this problem. This is the informal phase of development, while we are trying to understand what to do. Once we have an idea, we start designing functions and testing them.
A problem solved, a script built using function definitions

\[
t := \frac{d}{\text{speed}}
\]

12.42742384 (8.40)

\[\text{convert}(t, \text{units};'\text{hours}', '\text{minutes}')\]

745.6454304 (8.41)

It would pretty obvious how to make a script out of this to handle any problem of the form: A car travels \textit{speed} miles per hour. How many minutes does it take to travel \textit{distance} kilometers... With a few user defined functions, we can get the answer with less typing/cutting/pasting

\[
d\text{Convert} := (\text{distance}) \rightarrow \text{convert}(\text{distance}, \text{units}, \text{kilometers, miles})
\]

\[
\text{distance} \rightarrow \text{convert}(\text{distance}, \text{units}, \text{kilometers, miles})
\]

\[
d\text{Convert}(900)
\]

\[
\frac{781250}{1397}
\]

(8.43)

\[
t\text{Calc} := (d, \text{speed}) \rightarrow \frac{d}{\text{speed}}
\]

\[
(d, \text{speed}) \rightarrow \frac{d}{\text{speed}}
\]

(8.44)

\[
t\text{Calc}(1.418, 45)
\]

\[
\frac{156250}{12573}
\]

(8.45)

\[
t\text{Conv} := (t) \rightarrow \text{convert}(t, \text{units}; '\text{hours}', '\text{minutes}')
\]

\[
t \rightarrow \text{convert}(t, \text{units}; '\text{hours}', '\text{minutes}')
\]

(8.46)

\[
t\text{Conv}(1.415)
\]

\[
\frac{312500}{4191}
\]

(8.47)

\[
solveIt := (\text{speed, distance})
\]

\[
\rightarrow t\text{Conv}(t\text{Calc}(d\text{Convert}(\text{distance}), \text{speed}))
\]

\[
(\text{speed, distance})
\]

\[
\rightarrow t\text{Conv}(t\text{Calc}(d\text{Convert}(\text{distance}), \text{speed}))
\]

(8.48)

\[
solveIt(45, 900.0)
\]

\[
745.6454304
\]

(8.49)

The first step was to convert the distance from kilometers to miles. We create a function that does it.

We test it to the result that we got in the script above and see that it agrees.

The next step was to calculate the time (in hours) from the distance in miles and the speed in mph.

The test shows that \textit{tCalc} seems to be built correctly.

The third step was to convert the time from hours to minutes.

The test of this step agrees with what the script does, too.

The solution function chains together the three functions we've developed. This concludes the development and testing. We present a script and several solved problems in the figure below.
The above table showed the thinking behind the design and testing of the multi-step calculation. However, in "what we would hand in", we don't show the testing or the initial script, just the definition of the functions, and then the repeated invocation of the "solution function". Define the functions once, then invoke it repeatedly. This eliminates the need for repeated cutting/pasting/selection for execution.

<table>
<thead>
<tr>
<th>Solving multiple versions of a problem through functions</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Begin function definitions</strong></td>
</tr>
<tr>
<td>$d_{\text{Convert}} := (\text{distance}) \rightarrow \text{convert}(\text{distance}, \text{units}, \text{kilometers}, \text{miles})$</td>
</tr>
<tr>
<td>( \quad \text{distance} \rightarrow \text{convert}(\text{distance}, \text{units}, \text{kilometers}, \text{miles}) ) \hspace{1cm} (8.50)</td>
</tr>
<tr>
<td>$t_{\text{Calc}} := (d, \text{speed}) \rightarrow \frac{d}{\text{speed}}$</td>
</tr>
<tr>
<td>( \quad (d, \text{speed}) \rightarrow \frac{d}{\text{speed}} ) \hspace{1cm} (8.51)</td>
</tr>
<tr>
<td>$t_{\text{Conv}} := (t) \rightarrow \text{convert}(t, \text{units}, 'hours', 'minutes')$</td>
</tr>
<tr>
<td>( \quad t \rightarrow \text{convert}(t, \text{units}, 'hours', 'minutes') ) \hspace{1cm} (8.52)</td>
</tr>
<tr>
<td>$\text{travelSoln} := (\text{speed}, \text{distance}) \rightarrow t_{\text{Conv}}(t_{\text{Calc}}(d_{\text{Convert}}(\text{distance}), \text{speed}))$</td>
</tr>
<tr>
<td>( \quad (\text{speed}, \text{distance}) \rightarrow t_{\text{Conv}}(t_{\text{Calc}}(d_{\text{Convert}}(\text{distance}), \text{speed})) ) \hspace{1cm} (8.53)</td>
</tr>
<tr>
<td><strong>End of function definitions</strong></td>
</tr>
<tr>
<td><strong>Problem version A</strong></td>
</tr>
<tr>
<td>A car travels at 45 miles per hour. How many minutes does it take to travel 900 kilometers?</td>
</tr>
<tr>
<td>$\text{travelSoln}(45, 900.0)$ \hspace{1cm} 745.6454304 \hspace{1cm} (8.54)</td>
</tr>
<tr>
<td><strong>Problem version B</strong></td>
</tr>
<tr>
<td>A car travels at 45 miles per hour. How many minutes does it take to travel 452 kilometers?</td>
</tr>
<tr>
<td>$\text{travelSoln}(45, 452.0)$ \hspace{1cm} 374.4797052 \hspace{1cm} (8.55)</td>
</tr>
<tr>
<td><strong>Problem version C</strong></td>
</tr>
<tr>
<td>A car travels at 65 miles per hour. How many minutes does it take to travel 1500 kilometers?</td>
</tr>
<tr>
<td>$\text{travelSoln}(65, 1500.0)$ \hspace{1cm} 860.3601126 \hspace{1cm} (8.56)</td>
</tr>
</tbody>
</table>

The \text{travelSoln} function indicates the parameters of the script, \text{speed} and \text{distance}. In setting up the solution method as a function, we lose some intelligibility because of the "inside out" style of following the operation of daisy-chained function composition.

However, we gain convenience using the script. Using this form makes it easier to see several solutions, because you don't have to wade through all the lines of script that work out the answer, just a single line setting up the values of the parameters and printing out the answer.
8.5 Inputs and outputs to user-defined functions don't have to be numbers

Although you don't see much mention of this in mathematics texts, it is fairly common while programming to define and use functions that take inputs and produce outputs that are not numbers. For example, if we have a list L of numbers, we can create a function that takes a list as input and produces the average of all the numbers as its output.

Table 8.3: A function that takes a list as its input

<table>
<thead>
<tr>
<th>A function that takes a list as its input</th>
<th>Commentary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Example</td>
<td>nops(L)</td>
</tr>
</tbody>
</table>
| average(L) | \[
\frac{\sum_{i=1}^{\text{nops}(L)} L[i]}{\text{nops}(L)}
\] (8.57) |
| average([5, 7, -3, 2, 6]) | \[
\frac{17}{5}
\] (8.58) |
| average([5, 7, -3, 2, 6, 2, 2, 6]) | 3.375000000 (8.59) |

In analyzing mathematical models as we have been doing, it is also useful to produce abbreviations for common combinations of plot options by creating a function that produces a plot as its result.

Table 8.4: A function that returns a plot as its output, rather than a number

<table>
<thead>
<tr>
<th>Problem</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>A function that returns a plot as its output, rather than a number</td>
<td>A function that returns a plot is useful for visualizing data. It can be created by defining a function that takes the data as input and produces a plot as its output.</td>
</tr>
</tbody>
</table>
A function that returns a plot as its output, rather than a number

We are given two lists of data, \( pData \), and \( tData \). Plot \( pData \) as a function of \( tData \), and vice versa.

**Solution**

Build a function that becomes an abbreviation for the operations in the plot. Provide a third argument that is the string for the color.

\[
PlotI := (xData, yData, c, L) \rightarrow plot(xData, yData, style = point, color = c, labels = L)
\]

\[
PlotI := (xData, yData, c, L) \rightarrow plot(xData, yData, style = point, color = c, labels = L)
\] (8.60)

\[
pData := [134.2, 142.5, 155.0, 159.8, 171.1, 184.2];
\]

\[
pData := [134.2, 142.5, 155.0, 159.8, 171.1, 184.2]
\] (8.61)

\[
tData := [0, 20.1, 39.8, 60.0, 79.9, 100.3];
\]

\[
tData := [0, 20.1, 39.8, 60.0, 79.9, 100.3]
\] (8.62)

Plot pressure versus temperature, in red. Note: there seems to be a bug in Maple that suppresses the printing of the horizontal axis label.

\[
PlotI(pData, tData, "red", ["pressure", "temperature"])
\]

Plot temperature versus pressure, in red. Copying and pasting the invocation of the \( PlotI \) function we defined is easier than changing the insides of the original plot operation.

\[
PlotI(tData, pData, "blue", ["temperature", "pressure"])
\]
It is possible to return a list or sequence as a result of a function. Such a function can be put in a chain.

Table 8.5: A problem solved with a function that outputs a sequence of two numbers

<table>
<thead>
<tr>
<th>Problem A</th>
<th>A right triangle has a hypotenuse of length 5 feet. The angle between it and one of its sides is 10 degrees. What is the length of the perimeter?</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>We first build a function that computes the two sides of the right triangle and returns the two values as a sequence. We have to convert degrees into radians in order to do this because the Maple trig functions all use radians.</td>
</tr>
</tbody>
</table>
|           | \[
sideSide := \text{hypo, angle} \rightarrow \text{hypo sin(convert(angle degrees, radians)), hypo cos(convert(angle degrees, radians))}
\]
|           | \[
\text{hypo, angle} \rightarrow \text{hypo sin(convert(angle degrees, radians)), hypo cos(convert(angle degrees, radians))}
\] (8.63) |
|           | Let's test the sideSide function. |
|           | \[
sideSide(5, 10.0)
\] |
|           | \[
5 \sin(0.05555555556 \pi), 5 \cos(0.05555555556 \pi)
\] (8.64) |
|           | Now, develop a function that takes a sequence of three numbers and adds them together. |
|           | \[
sumSides := (a, b, c) \rightarrow a + b + c
\]
|           | \[
(a, b, c) \rightarrow a + b + c
\] (8.65) |
|           | By chaining together the output of sideSide and making it part of the input of sumSides, we can get the whole computation done in one function. |
A problem solved with a function that outputs a sequence of two numbers

\[
\text{perimeter} := (\text{hypo}, \text{angle}) \rightarrow \text{sumSides}(\text{sideSide}(\text{hypo}, \text{angle}), \text{hypo})
\]

\[
(\text{hypo}, \text{angle}) \rightarrow \text{sumSides}(\text{sideSide}(\text{hypo}, \text{angle}), \text{hypo})
\]  

(8.66)

\[
\text{perimeter}(5, 10)
\]

\[
5 \sin \left( \frac{1}{18} \pi \right) + 5 \cos \left( \frac{1}{18} \pi \right) + 5
\]

(8.67)

at 5 digits

\[
10.792
\]

(8.68)

Problem B

A right triangle has a hypotenuse of length 10 feet. The angle between it and one of its sides is 42 degrees. What is the length of the perimeter?

\[
\text{evalf}(\text{perimeter}(10, 42))
\]

\[
24.12275432
\]

(8.69)

Once we have done the work to design and test the functions out on a problem, we can present a script that can solve several different versions of the problem:

### Solving several versions of a function with function definitions

**Begin function definitions**

A function that computes the two sides of a right triangle given the angle and the length of the hypotenuse

\[
\text{sideSide} := (\text{hypo}, \text{angle}) \rightarrow (\text{hypo} \cdot \sin(\text{convert}(\text{angle} \cdot \text{degrees}, \text{radians})), \text{hypo} \cdot \cos(\text{convert}(\text{angle} \cdot \text{degrees}, \text{radians})))
\]

\[
(\text{hypo}, \text{angle}) \rightarrow (\text{hypo} \sin(\text{convert}(\text{angle} \text{degrees}, \text{radians})), \text{hypo} \cos(\text{convert}(\text{angle} \text{degrees}, \text{radians})))
\]  

(8.70)

A function that takes a sequence of three numbers and adds them together.

\[
\text{sumSides} := (\text{a}, \text{b}, \text{c}) \rightarrow \text{a} + \text{b} + \text{c}
\]

\[
(\text{a}, \text{b}, \text{c}) \rightarrow \text{a} + \text{b} + \text{c}
\]  

(8.71)

Compute the perimeter by summing the three sides.

\[
\text{perimeter} := (\text{hypo}, \text{angle}) \rightarrow \text{sumSides}(\text{sideSide}(\text{hypo}, \text{angle}), \text{hypo})
\]

\[
(\text{hypo}, \text{angle}) \rightarrow \text{sumSides}(\text{sideSide}(\text{hypo}, \text{angle}), \text{hypo})
\]  

(8.72)

**End function definitions**

**Problem A Solution**

A right triangle has a hypotenuse of length 5 feet. The angle between it and one of its sides is 10 degrees. Approximately, what is the length of the perimeter in feet?

\[
\text{evalf}(\text{perimeter}(5, 10))
\]
### 8.6 Chapter Summary

#### Function design

<table>
<thead>
<tr>
<th>Designing functions from context</th>
<th>a) What will the inputs be?</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>b) What will the output be?</td>
</tr>
<tr>
<td></td>
<td>c) How do we calculate the output from the inputs?</td>
</tr>
</tbody>
</table>

#### Function composition

\[
A := (x, y) \rightarrow \frac{1}{x} + 3 \cdot x^2 + 3 \cdot y
\]
\[
B := (x, y) \rightarrow \frac{3}{A(x, y)}
\]

\[
B(3, 1) = \frac{9}{253} \quad (8.77)
\]

#### Unit conversion

- Units can be converted directly into compatible units.
- Compound units expressed in ratio form can also be converted into compatible compound units.
- Converting between incompatible units will generate an error message.
- If we create a variable with the same name as a unit, trying to convert using the variable name will throw an error.

\[
convert(1, \text{units}, \text{inch}, \text{meter})
\]
\[
\frac{127}{5000} \quad (8.78)
\]

\[
convert\left(15 \text{\,0 miles per hour} \div \text{second}\right)
\]
\[
6.705600000 \quad (8.79)
\]

\[
convert(3, \text{units}, \text{days, miles})
\]

Error, (in convert/units) unable to convert ‘d’ to ‘mi’

\[
seconds := 5
\]
\[
5 \quad (8.80)
\]
### Unit conversion

To avoid this, we can use single quotes around the unit names to specify using the unit string, as opposed to the variable value.

```latex
\text{convert}(3, \text{ units, days, seconds})
```

Error, (in convert/units) unable to convert 'd' to 'S'

```latex
\text{convert}(3, \text{ units,'days','seconds'})
```

\[ 259200 \quad (8.81) \]

We can use Maple's built-in unit converter to convert units using drop-down menus.

This tool is located in Tools>Assistants[Unit Calculator]

### Non-number inputs and outputs of a function

Using a list of numbers as an input

\[
\text{average} := L \rightarrow \frac{\sum_{i=1}^{\text{nops}(L)} L[i]}{\text{nops}(L)}
\]

\[
\text{average}([5, 7, -3, 2, 6])
\]

\[
\frac{17}{5} \quad (8.83)
\]

Returning a plot instead of a number

\[
\text{PlotIt} := (x\text{Data}, y\text{Data}, c, L) \rightarrow \text{plot}(x\text{Data}, y\text{Data}, \text{style} = \text{point}, \color = c, \text{labels} = L)
\]

\[
(x\text{Data}, y\text{Data}, c, L) \rightarrow \text{plot}(x\text{Data}, y\text{Data}, \text{style} = \text{point}, \color = c, \text{labels} = L)
\]

\[(8.84)\]

\[
\text{PlotIt}([1, 2, 3], [2.1, 2.1], 'blue', ['x', 'y'])
\]

![Plot](image)

Returning a sequence of numbers instead of a single number

\[
\text{sideSide} := (\text{hypo}, \text{angle}) \rightarrow (\text{hypo}\cdot\sin(\text{convert(angle\cdotdegrees, radians)}), \text{hypo}\cdot\cos(\text{convert(angle\cdotdegrees, radians)}))
\]

\[(8.85)\]
| Non-number inputs and outputs of a function | $sideSide(1, 45)$ | $\frac{1}{2} \sqrt{2}, \frac{1}{2} \sqrt{2}$ | (8.86) |