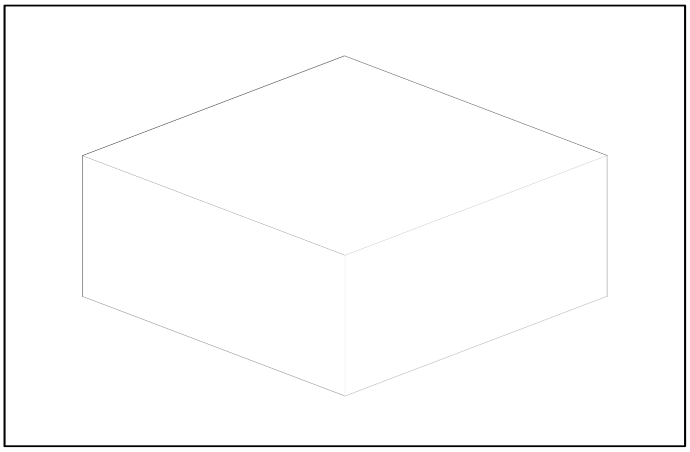
1 Building a Block Factory: Creating a Maple Procedure

In this section we want to begin to learn how to define new **Maple words.** After a little bit of effort, you will find that it is as natural as driving a car (and not nearly as dangerous, because there is no mistake in Maple that can't be fixed.) The subject is blocks, also known as cubes.

1.1 Start with a single cube.

If we are to construct more complex figures out of cubes and manipulate them in more interesting ways we need a more efficient means of producing the rectangular boxes with which to build them. It would be nice if there were a Maple word which instructed the program to make, say, a cube of side 1 and place it at a location of our choice. In the previous chapter we did all of the work that such a word would do. We constructed the basic cube and placed it in the desired location by specifying a 'shift'. The steps to create a simple, colorless cube might be:

```
front:=[v1,v2,v3,v4]:  back:=[v5,v6,v7,v8]:
left:=[v4,v8,v5,v1]:
right:=[v3,v2,v6,v7]:  bottom:=[v3,v4,v8,v7]:  top:=[v1,v2,v6,v5]:
box:=[front,back,left,right,top,bottom]:
shift:=[0,0,0]:
v1:=[0,0,0]+shift:  v2:=[1,0,0]+shift:  v3:= [1,1,0]+shift:
v4:=[0,1,0]+shift:
v5:=[0,0,1]+shift:  v6:=[1,0,1]+shift:  v7:=[1,1,1]+shift:
v8:=[0,1,1]+shift:
plots[polygonplot3d](box);
```



If we wanted the cube placed at any particular place in space all we have to do is change the value of "shift". If we wanted to retain the previous cube then instead of simply having **plots[polygonplot3d]** immediately draw the picture we assigned the plot structucture generated to a word, say "cube1". Then we could change the shift and the name, say to "cube2" and have plots[display] show us our cubes separately or individually as we wish. What we would like to do is have a Maple word, say "cube" which would take in the shift and produce the plot structure for the corresponding cube. We want the command "cube([0,0,0])" to produce the results of the previous set of commands and would want "cube([0,0,1])"

to produce the results of simply changing "shift" to "[0,0,1]" in the same list of commands. That is, all we want is a way to vary

the value of "shift " in this set of commands. Maple does this with **procedures.** The form of a Maple procedure is simple:

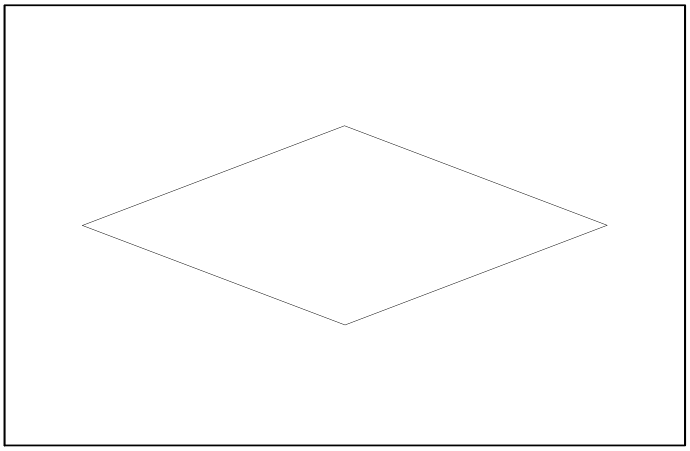
ProcedureName:= proc(input_variable)

Maple commands in which "input_variable" is treated as input.

end

For instance suppose we want a Maple word which always returns a picture of a unit square of whichever color we specify. We need a choice for this word, the ProcedureName. Any choice will work provided its not "reserved" by Maple. For instance we might choose "colorsquare". We need a name for the procedure variable. Since we are going to vary color we choose something like "clr" since we have seen that "color" is a word already defined in Maple. Finally we need to know the Maple word which will produce a square of a given color. We know that the following will work:

```
plots[polygonplot3d]([[0,0,0],[1,0,0],[1,1,0]
,[0,1,0]], color=red);
```



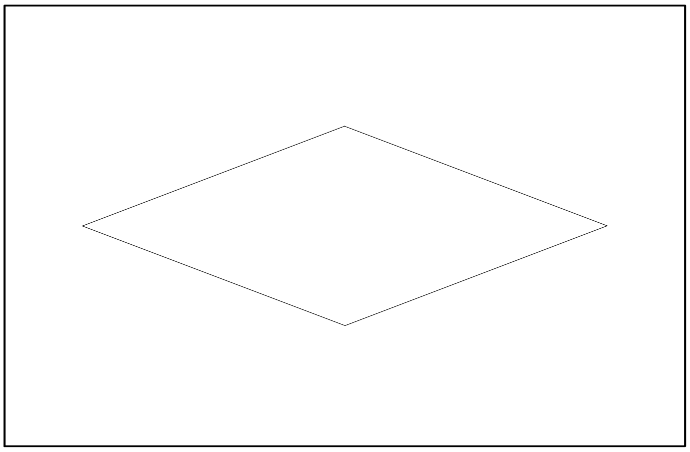
Thus following the above format we create a new word:

```
colorsquare:=proc(clr)
plots[polygonplot3d]([[0,0,0],[1,0,0],[1,1,0],[0,1,0]], color=clr)
end;
```

$$colorsquare := \mathrm{proc}(clr)\ plots_{polygonplot3d}([[0,0,0],[1,0,0],[1,1,0],[0,1,0]],\ color = clr)\ \mathrm{end}$$

We can test the word out right away.

```
colorsquare(blue);
```



This word only colors the boundary of the square. To find out how to color the interior of the square, read on.

Creating our 'cube' word is no more difficult and is done essentially by cutting and pasting the code that does one case, changing the specific values in the code to the variable quantities for the procedure and 'wrapping' this in the 'proc()...end;' format.

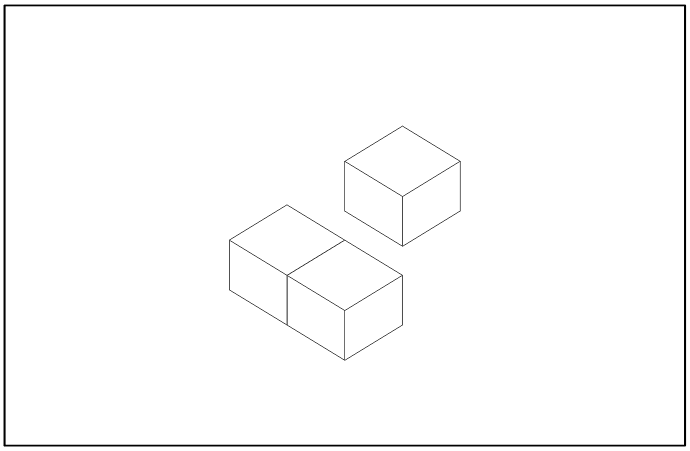
```
cube:=proc(shift)
v1:=[0,0,0]+shift:  v2:=[1,0,0]+shift:  v3:= [1,1,0]+shift:
v4:=[0,1,0]+shift:
v5:=[0,0,1]+shift:  v6:=[1,0,1]+shift:  v7:=[1,1,1]+shift:
v8:=[0,1,1]+shift:
front:=[v1,v2,v3,v4]:  back:=[v5,v6,v7,v8]:  left:=[v4,v8,v5,v1]:
right:=[v3,v2,v6,v7]:  bottom:=[v3,v4,v8,v7]:  top:=[v1,v2,v6,v5]:
box:=[front,back,left,right,top,bottom]:
plots[polygonplot3d](box);
end:
```

For the moment pay no attention to the "warning" messages, we will clean that up shortly. Right not what we need to see is that we do indeed have a "cube" word. The idea is always to get something working and improve it. We can give names to some particular cubes and then display them all together.

```
joe:=cube([1,2,4]);
sam:=cube([1,0,1]);
sally:=cube([1,1,1]):
plots[display]([joe,sam,sally],color=red,scaling=constrained);
```

$$
\begin{aligned}
joe := &\mathbf{PLOT3D(POLYGONS(}[[1., 2., 4.], [2., 2., 4.], [2., 3., 4.], [1., 3., 4.]],\\
&[[1., 2., 5.], [2., 2., 5.], [2., 3., 5.], [1., 3., 5.]],\\
&[[1., 3., 4.], [1., 3., 5.], [1., 2., 5.], [1., 2., 4.]],\\
&[[2., 3., 4.], [2., 2., 4.], [2., 2., 5.], [2., 3., 5.]],\\
&[[1., 2., 4.], [2., 2., 4.], [2., 2., 5.], [1., 2., 5.]],\\
&[[2., 3., 4.], [1., 3., 4.], [1., 3., 5.], [2., 3., 5.]]))
\end{aligned}
$$

$sam :=$ **PLOT3D(POLYGONS([[1., 0, 1.], [2., 0, 1.], [2., 1., 1.], [1., 1., 1.]], [[1., 0, 2.], [2., 0, 2.], [2., 1., 2.], [1., 1., 2.]], [[1., 1., 1.], [1., 1., 2.], [1., 0, 2.], [1., 0, 1.]], [[2., 1., 1.], [2., 0, 1.], [2., 0, 2.], [2., 1., 2.]], [[1., 0, 1.], [2., 0, 1.], [2., 0, 2.], [1., 0, 2.]], [[2., 1., 1.], [1., 1., 1.], [1., 1., 2.], [2., 1., 2.]]))**



When a word which is not a Maple or user-defined word is used within a Maple procedure it is allowed to have meaning only within the procedure. It is used only locally, hence the "local variable" designation. This allows words to be used within procedures that actually have an external meaning. This is very useful in the development of procedures. For instance, our objective in developing "cube" to no longer have to think about its details when we need a cube in the future. Suppose we were to use it in the future as part of a project in which we have also used the word "top" to mean the amount of fuel in a tank when it is full. Now having forgotten all about the internal workings of "cube" we issue the command "cube([0,0,0])". Note that within "cube" there is a command "top:=[v1,v2,v6,v5]". If "top" in "cube" were not restricted to have meaning only within that procedure its invovation would have changed the "top" we were thinking of from a number to a four element list. This could cause all kinds of problems.

The default is that Maple makes all such variables local variables and tells us about it with that message. If we want to tell the program in advance that these are to be local variables we provide it the a sequence of those variables in the code defining the procedure.

There is a corresponding idea of **global variable,** which is what we would use if we really did want "cube" to modify "top". Here is the way we would have entered "cube" including the local variables.

```
cube := proc (shift)
local v1, v2, v3, v4, v5, v6, v7, v8, front, back, left, right,
bottom, top, box;
v1 := [0, 0, 0]+shift; v2 := [1, 0, 0]+shift;
v3 := [1, 1, 0]+shift; v4 := [0, 1, 0]+shift;
v5 := [0, 0, 1]+shift; v6 := [1, 0, 1]+shift;
v7 := [1, 1, 1]+shift; v8 := [0, 1, 1]+shift;
front := [v1, v2, v3, v4]; back := [v5, v6, v7, v8];
left := [v4, v8, v5, v1]; right := [v3, v2, v6, v7];
bottom := [v3, v4, v8, v7]; top := [v1, v2, v6, v5];
box := [front, back, left, right, top, bottom];
```

```
plots[polygonplot3d](box)
end:
```

We will use this **code** again in next worksheet. The word code is used throughout this text to refer to sequences of Maple statements which may be part of a procedure definition.

Now that we have a basic version of "cube" we can add features. Suppose we want to specify the style and color of the cube as well as the shift. This is easy as Maple procedures are not restricted to a single variable. It is a good idea to make a single improvement at a time, though. This makes finding errors much easier. Lets specify the style first. All we need to do is copy the code over. We don't go back and change our only copy of a piece of working code. We copy the working version, paste it in and (ususally) change the name at least temporarily

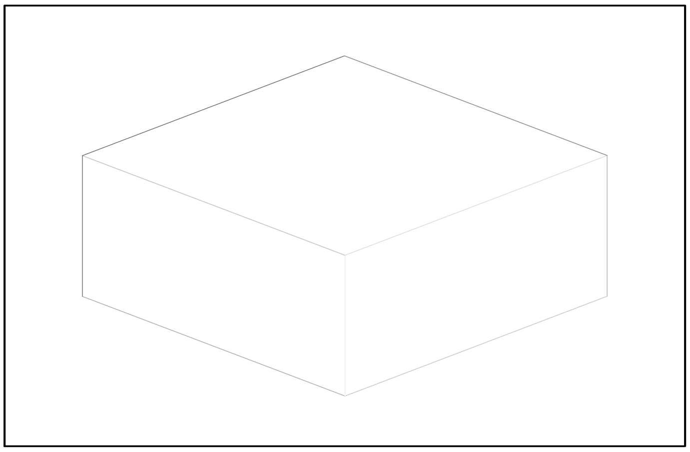
```
tempcube := proc (shift)
local v1, v2, v3, v4, v5, v6, v7, v8, front, back, left, right,
bottom, top, box;
v1 := [0, 0, 0]+shift; v2 := [1, 0, 0]+shift;
v3 := [1, 1, 0]+shift; v4 := [0, 1, 0]+shift;
v5 := [0, 0, 1]+shift; v6 := [1, 0, 1]+shift;
v7 := [1, 1, 1]+shift; v8 := [0, 1, 1]+shift;
front := [v1, v2, v3, v4]; back := [v5, v6, v7, v8];
left := [v4, v8, v5, v1]; right := [v3, v2, v6, v7];
bottom := [v3, v4, v8, v7]; top := [v1, v2, v6, v5];
box := [front, back, left, right, top, bottom];
plots[polygonplot3d](box)
end:
```

This is simply the original code with a different name. Now modify it to take both a shift and a color. We have indicated the changes in upper case in the following copy. In this case we simply add a variable "clr" and the option "color=clr" in the plots[polygonplot3d] command.

```
tmpcube := proc (shift,CLR)
local v1, v2, v3, v4, v5, v6, v7, v8, front, back, left, right,
bottom, top, box;
v1 := [0, 0, 0]+shift; v2 := [1, 0, 0]+shift;
v3 := [1, 1, 0]+shift; v4 := [0, 1, 0]+shift;
v5 := [0, 0, 1]+shift; v6 := [1, 0, 1]+shift;
v7 := [1, 1, 1]+shift; v8 := [0, 1, 1]+shift;
front := [v1, v2, v3, v4]; back := [v5, v6, v7, v8];
left := [v4, v8, v5, v1]; right := [v3, v2, v6, v7];
bottom := [v3, v4, v8, v7]; top := [v1, v2, v6, v5];
box := [front, back, left, right, top, bottom];
plots[polygonplot3d](box,color=CLR)
end:
```

Now test this change right away.

```
tempcube ([0,0,0],red);
```

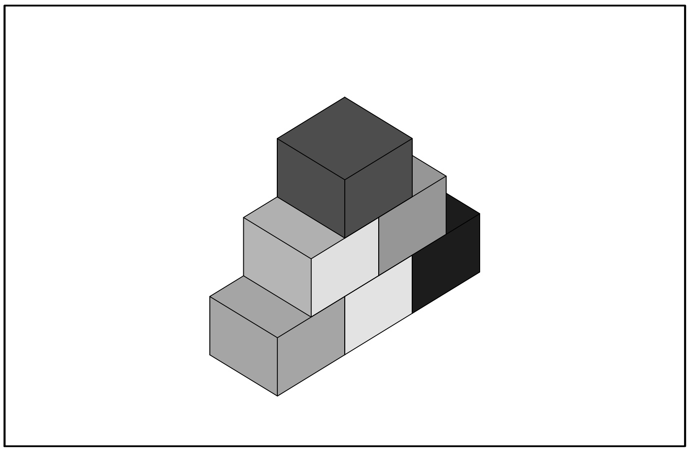


Now we can go another step and put the style choice in the inputs.

```
tempcube2 := proc (shift,CLR,STY)
local v1, v2, v3, v4, v5, v6, v7, v8, front, back, left, right,
bottom, top, box;
v1 := [0, 0, 0]+shift; v2 := [1, 0, 0]+shift;
v3 := [1, 1, 0]+shift; v4 := [0, 1, 0]+shift;
v5 := [0, 0, 1]+shift; v6 := [1, 0, 1]+shift;
v7 := [1, 1, 1]+shift; v8 := [0, 1, 1]+shift;
front := [v1, v2, v3, v4]; back := [v5, v6, v7, v8];
left := [v4, v8, v5, v1]; right := [v3, v2, v6, v7];
bottom := [v3, v4, v8, v7]; top := [v1, v2, v6, v5];
box := [front, back, left, right, top, bottom];
plots[polygonplot3d](box,color=CLR,style=STY)
end:
```

Now we can play with colored cubes:

```
cube1:=tempcube2([0,0,0],blue,patch):
cube2:=tempcube2([1,0,0],yellow ,patch):
cube3:=tempcube2([2,0,0],tan ,patch):
cube4:=tempcube2([1/2,0,1],green ,patch):
cube5:=tempcube2([3/2,0,1],purple ,patch):
cube6:=tempcube2([1,0,2],red,patch):
plots[display]([ cube1, cube2, cube3, cube4, cube5, cube6 ],
scaling = constrained);
```



1.2 Another Example of how to 'procedurize' a sequence of commands

It is very important to learn the process of organizing sequences of Maple commands which solve a specific problem into a procedure for doing a general problem of that form. For example. The following Maple code calculates the equation of the line though the two points [1,3] and [-5,12].

```
eqn:= y=slope*x+yintercept;
```

$$eqn := y = slope\, x + yintercept$$

```
slope:= (12 - 3)/(-5 - 1);
```

$$slope := \frac{-3}{2}$$

```
yintercept:=3-slope*1;
```

$$yintercept := \frac{9}{2}$$

```
eqn;
```

$$y = -\frac{3}{2}\,x + \frac{9}{2}$$

We want to modify this to take in the x and y coordinates of any two points [x1,y1] and [x2,y2] and to return the equation of the line through them. Once we see that we should think of [1,3] as the particular [x1,y1] and [-5,12] as the particular [x2,y2] we can readily wrap this into a procedure with a name such as "lineequ". Note that we are ignoring for the moment the possibility that x1=x2 in which case Maple will complain about division by zero. We'll learn to take care of such things later.

```
lineequ:=proc(x1,y1,x2,y2)
local slope, intercept, eqn;
eqn:= y=slope*x+intercept;
slope:= (y2 - y1)/(x2 - x1);
intercept:= y1 - slope*x1;
y=slope*x+intercept;
end:
```

Then to calculate the equation of the line passing through [1,3] and [-5,12] one can execute the command

```
lineequ( 1,3, -5,12);
```

$$y = -\frac{3}{2}x + \frac{9}{2}$$

1.3 EXERCISES

1.3.1 Exercise: *Modifty the procedure lineqn above so its inputs are a two points in the plane rather than a four numbers.*

1.3.2 Exercise : Modify the tempcube2 procedure to make a word which makes cubes which have white tops and bottoms and whose sides are any color we specify.

1.3.3 Exercise: Modify the following Maple commands find the point of intersection of two specific lines in the plane from their equations, each of the form $y = mx + b$. Make this code into a procedure intersect(m1,b1,m2,b2) which finds the point of intersection of the lines with equations $y = m1x + b1$ and $y = m2x + b2$. (remark: We are not worrying at this point about such niceties as whether the two lines are parallel or not. We will learn to check for this sort of thing later). Test your word on several lines.

```
m1:=10; b1:=13; m2:=7; b2:=25;
solve({y=m1*x+b1,y=m2*x+b2},{x,y});
```

$$m1 := 10$$

$$b1 := 13$$

$$m2 := 7$$

$$b2 := 25$$

$$\{y = 53,\ x = 4\}$$

1.3.4 Exercise : The Maple command sum(i,i=1..4781); adds up the first 4781 positive integers. Adapt it to make a Maple word addpowers(n,m) which returns the sum of the nth powers of the integers from 1 to m. Use your word to answer the following question: Which is larger, the sum of the 4th powers of the first 100 positive integers, or the sum of the third powers of the first 300 integers?

1.3.5 Exercise: *What does the following Maple procedure do? If a word is unfamiliar, use the Maple Help to find out about it.*

```
ndigits:=proc(n) local tmp:
Digits:=n; tmp:=evalf(Pi); Digits:=10:  tmp; end;
```

$$ndigits := \mathrm{proc}(n)\ \mathrm{local}\, tmp;\ Digits := n\,;\ tmp := \mathrm{evalf}(\pi)\,;\ Digits := 10\,;\ tmp\ \mathrm{end}$$

1.3.6 Exercise *What does the following Maple procedure do? If a word is unfamiliar, use the Maple Help to find out about it.*

```
dig:=proc(n) floor(10*frac(10^(n-1)*Pi))
end;
```

$$dig := \mathrm{proc}(n)\ \mathrm{floor}(10 \times \mathrm{frac}(10^{(n-1)} \times \pi))\ \mathrm{end}$$

1.3.7 Exercise: The following is a Maple procedure which almost works. It should input a list of four numbers and make a bar graph representation of that data with the bar representing the first number having as it base the line segment [0,1] on the x-axis, the bar representing the second number should have as its base the segment [1,2], etc. The color of the first bar should be red, the second green, the third yellow, and the fourth blue.

```
barit:=proc(l) local bar1,bar2,bar3, bar4;
bar1:=plots[polygonplot]( [[2,0],[4,0],
[4,l[3]],[2,l[3]]],color=green):
bar2:=plots[polygonplot]( [[0,0],[1/2,0],
[1/2,l[4]],[0,l[4]]],style=patch,color=red):
bar3:=plots[polygonplot]( [[3,1],[5,1], [5,l[1]],[3,l[1]]],
style=LINE,color=blue,thickness=3):
bar4:=plots[polygonplot]( [[5,0],[6,0],
[6,-l[2]],[5,-l[2]]],style=patch,color=yellow):
plots[display]([bar1,bar2,bar3,bar4]); end;
```

Here is barit([5,2,7,-4])

```
\textbf{Exercise}: The following Maple procedure is supposed to input
three points in the plane and check to see if they are collinear. Read
the instructions carefully and explain exactly what the procedure is
doing.
```

```
checkit:=proc(P,Q,R) local slope, intercept,
eqn;
eqn:= y=slope*x+intercept;
slope:= (Q[2] - P[2])/(Q[1] - P[1]);
intercept:= Q[2] - slope*Q[1];
R[2]=slope*R[1]+intercept
end;
```

1.3.8 Exercise: *The following procedure inputs a list of four points in the plane and does a calculation which hopes to tell whether the four points are the vertices of a square. Draw a diagram of a quadralateral, label the vertices in accordance with the notation used by the procedure and explain with reference to the geometry of the figure exactly what the procedure is doing. Explain why its output can or can not tell one whether the figure is in fact a square*

Recall that if P is a list of things that are themselves lists then P[3,7] denotes the seventh element of the third member of the list P.

```
isit:=proc( P) local tmp1,tmp2:
```

```
tmp1:= ((P[1,1]-P[3,1])^2 + ( P[1,2]-P[3,2])^2) - (
(P[2,1]-P[4,1])^2 + (
P[2,2]-P[4,2])^2 );
tmp2:= ((P[1,1]-P[2,1])^2 + ( P[1,2]-P[2,2])^2) - ( (P[2,1]-P[3,1])^2
+ (
P[2,2]-P[3,2])^2 );
tmp1^2+tmp2^2;
end:
```