Maple Programming

K. Cooper

2012
We can make piecewise functions using the *piecewise* function.
Piecwise

We can make piecewise functions using the *piecewise* function.

\[ \text{piecewise}(\text{condition1}, \text{expression1}, \text{condition2}, \text{expression2}, \ldots) \]

If \( \text{condition1} \) is true, then \( \text{expression1} \) is the result of the function. If \( \text{condition1} \) is false and \( \text{condition2} \) is true, then \( \text{expression2} \) is the result; and so on.
We can make piecewise functions using the `piecewise` function.

\[
piecewise(condition1, expression1, condition2, expression2, \ldots)
\]
If `condition1` is true, then `expression1` is the result of the function. If `condition1` is false and `condition2` is true, then `expression2` is the result; and so on.

\[
f := x \mapsto \text{piecewise}(x < 0, 0, x \leq 1, x^2, x > 1, x^2 - 1);
\]
Piecewise

We can make piecewise functions using the `piecewise` function.

`piecewise(condition1, expression1, condition2, expression2, ...)`

If `condition1` is true, then `expression1` is the result of the function. If `condition1` is false and `condition2` is true, then `expression2` is the result; and so on.

\[ f := x \rightarrow \text{piecewise}(x < 0, 0, x \leq 1, x^2, x > 1, x^2 - 1); \]

\[ \text{plot}(f(x), x = -2..2); \]
We can make piecewise functions using the *piecewise* function.

\[
\text{piecewise}(\text{condition1}, \text{expression1}, \text{condition2}, \text{expression2}, \ldots)
\]

If \text{condition1} is true, then \text{expression1} is the result of the function. If \text{condition1} is false and \text{condition2} is true, then \text{expression2} is the result; and so on.

\[
f := x \rightarrow \text{piecewise}(x < 0, 0, x \leq 1, x^2, x \geq 1, x^2 - 1);
\]

\[
\text{plot}(f(x), x = -2..2, \text{discont} = \text{true});
\]
Making Lists

To create a sequential list, use the \texttt{seq()} command.
Making Lists

To create a sequential list, use the `seq()` command.

Given $h$ and $n$

\[ x := [\text{seq}(i \times h, i = 0 .. n)] \]  

\[ [0, h, 2h, \ldots, nh]. \]
Sometimes we need to do more complex tasks that require evaluation of many expressions, possibly repetitively.
Sometimes we need to do more complex tasks that require evaluation of many expressions, possibly repetitively.

```
name:=proc(arguments) local variables; ... end proc;
```

We can put any number of Maple commands we like where the ellipses are. The last object evaluated is returned as the result of the procedure.
Sometimes we need to do more complex tasks that require evaluation of many expressions, possibly repetitively.

```
name := proc(arguments) local variables; ... end proc;
```

We can put any number of Maple commands we like where the ellipses are. The last object evaluated is returned as the result of the procedure.

Procedures constitute a way to create new Maple commands. E.g.

```
quad := proc(x); x^2; end proc;
```

gives something with the same effect as `quad := x -> x^2;`
- The last object evaluated by the procedure is the result of the procedure.
- If you want to terminate the procedure early and send back an intermediate result, you can use `return`.
- Sometimes you can use `return` just to emphasize the result, or to force the result to be e.g. an entire list, rather than just the last element computed.
Procedures would be pointless if all we did was define little functions.
Procedures would be pointless if all we did was define little functions.

\[
\text{trapezoidalrule} := \text{proc}(\text{expr}, a, b, n) \text{ local } h, y, \text{total}; \\
h := (b-a)/n; \\
y := \text{seq}(i*h, i=0..n); \\
\text{total} := h*\text{add}(\text{eval}(\text{expr}, x=y[i]), i=1..n+1); \\
\text{total} := \text{total} - (1/2)*h*(\text{eval}(\text{expr}, x=y[1]) - \text{eval}(\text{expr}, x=y[n+1])); \\
\text{end proc;}
\]
Conditionals

```maple
if cond1 then expr1; elif cond2 then expr2; else expr3; end if;
```
if cond1 then expr1; elif cond2 then expr2; else expr3; end if;

h := x -> if x < 0 then
    0;
elif x < 1 then
    x^2;
else x^2 - 1;
end if;
Repetitive Operations

We can use \textit{sum()} or \textit{add()} to sum simple expressions, and we can use \textit{seq()} to create simple lists.
Repetitive Operations

We can use \texttt{sum()} or \texttt{add()} to sum simple expressions, and we can use \texttt{seq()} to create simple lists.

When that is not enough, we can use \texttt{for}

\begin{verbatim}
for var from start by step to end do exprs; end do;
\end{verbatim}
We can use \texttt{sum()} or \texttt{add()} to sum simple expressions, and we can use \texttt{seq()} to create simple lists.

When that is not enough, we can use \texttt{for}

\begin{verbatim}
for var from start by step to end do exprs; end do;
\end{verbatim}

Given \(h\) and \(n\);

\begin{verbatim}
for i from 0 to n do x[i] := i*h; end do;
\end{verbatim}
For loops

- You can put as many expressions as you like between `do` and `end do`.
- You can step by any increment you want, but the default is one.
- The actual action is to check at each iteration whether the counting variable is greater than the upper bound. If so, execution terminates, even if the expressions inside have never been evaluated.
You can also do loops for elements in a set.
For loops

You can also do loops for elements in a set.

for var in collection do exprs; end do;
For loops

You can also do loops for elements in a set.

```
for var in collection do exprs; end do;
i:=1; for x in {2,3,5,7,11} do y[i]:=x^2; i:=i+1; end do;
```
While loops

Use *for* loops when you know exactly how many times something needs to be iterated.
While loops

Use *for* loops when you know exactly how many times something needs to be iterated.

When you just need to repeat until some condition is true, use a *while* loop.
While loops

Use *for* loops when you know exactly how many times something needs to be iterated.

When you just need to repeat until some condition is true, use a *while* loop.

*while condition do exprs; end do;*
Use *for* loops when you know exactly how many times something needs to be iterated.

When you just need to repeat until some condition is true, use a *while* loop.

```
while condition do exprs; end do;
```

```
x := 10; while x > 1/10 do x := x/2; end do;
```
While loops

- Do not use while loops when you know beforehand how many iterations you need.
- Do not forget to change the variable you are testing – while loops are the easiest way to put the computer in an infinite repetition.
For... while

You can mix and match for and while loops.
For... while

You can mix and match for and while loops.

\[
\text{for var from start to end while condition do exprs; end do;}
\]
For... while

You can mix and match for and while loops.

```maple
for var from start to end while condition do exprs; end do;

for t in 2, 3, 5, 7 do
    if 2*floor((1/2)*t) = t then print(t, " is even"); end if;
end do;
```
Example

secdirmat:=proc(n) local A, i;
A:=Matrix(n,n);
for i from 1 to n-1 do
A[i,i] := -2;
A[i,i+1] := 1;
A[i+1,i] := 1;
end do;
A[n,n] := -2;
return A;
end proc;