Developing Large Codes Sets In Maple

- Develop in text files.
- Read these files into maple in one of two ways.
A file (that you should put in your home directory) that Maple will read and execute when you run. Useful things to put in it:

1. `kernelopts(printbytes=false)`:  
2. `interface(verboseproc=2)`:  
3. `plotsetup(maplet); or plotsetup(x11);`  
4. `interface(imaginaryunit=I)`: 
List Extension

BadList := proc(n)
local i, LL;

    LL := [];

    for i from 1 to n do
        LL := [ op(LL), i ];
    end do;

    return LL;
end proc:

[> TIME( BadList( 10^5 ) );
[ 66.401
TableList := proc(n)
    local i, LL;

    LL := table();

    for i from 1 to n do
        LL[i] := i;
    end do;

    return [ seq( LL[i], i=1..n ) ];
end proc:

[> TIME( TableList( 10^5 ) );
[ 0.215
(308 times faster).
A Maple list is an immutable array. So, each new list costs $O(i)$ where $i$ is the length of the list. Put this in a loop you get

$$\sum_{i=1}^{n} O(i) = O\left(\frac{n(n + 1)}{2}\right) = O(n^2)$$

and thus: bad list extension is quadratic.

Good list extension is linear.
Useful list shortcuts

```maple
L := [ seq( i, i=1..10) ];
L := [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
> nops( L );
10
> L[-1];
10
> L[3..5];
[3, 4, 5]
> L[2..-1];
[2, 3, 4, 5, 6, 7, 8, 9, 10]
> L[1..-2];
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Plus all the extra commands in ListTools.
Using The Right Function

```
BadFloor := proc( N )
local i, x;

for i from 1 to N do
    x := rand();
    floor( x );
end do;

end proc:

[> TIME( BadFloor( 10^7 ) );
[ 73.836
```
GoodFloor := proc(N)
local i, x;
    for i from 1 to N do
        x := rand();
        trunc(x);
    end do;
end proc:

> TIME(GoodFloor(10^7));
[ 38.308
(1.9274303 times faster).
floor will work (or try to work) on virtually anything you give to it. trunc is a simple C instruction.

**Others**

1. sum versus add
2. \( \div \) versus quo or rem
types

```maple
FooBar := proc(f::list(polynom(posint)), n::nonnegint)
    return 0;
end proc:

[> FooBar( [2-x], 3);
[ Error, invalid input: FooBar expects its 1st argument, f, to be of type list(polynom(posint)), but received [2-x]
[> FooBar( [2+x], -3);
[ Error, invalid input: FooBar expects its 2nd argument, n, to be of type nonnegint, but received -3
[> ? type;
```
\[
\begin{align*}
\text{Maple Tips, Tricks and Hacks} \\
\text{Paul Vrbik \quad University of Western Ontario} \\
\text{Maple Tips, Tricks and Hacks}
\end{align*}
\]

\[
[> \ f := \text{Matrix}( [[x^3 + 2*x - 4],[x - 2]] ); \\
[> \ \text{type}( f, \text{Matrix}[\text{polynom}] ); \\
[ true \\
[> \ \text{type}( f, \text{Matrix}[\text{polynom(integer)}] ); \\
[ true \\
[> \ \text{type}( f, \text{Matrix}[\text{polynom(posint)}] ); \\
[ false
\]
Remember Tables

```maple
FooBar := proc(n::posint)
local i;

    for i from 1 to 10^5 do
        i^i;
    end do;

    FooBar(n) := n;

    return n;

end proc:

[> Time( FooBar(3) );
[ 22.243
[> Time( FooBar(3) );
[ 0.
```

Paul Vrbik  University of Western Ontario  Maple Tips, Tricks and Hacks
Remember Tables

\[
\text{FACT} := n \rightarrow n \times \text{FACT}(n-1); \\
\text{FACT}(5); \\
\text{Error, (in FACT) too many levels of recursion} \\
\text{FACT}(0):=1; \\
\text{FACT}(5); \\
120
\]
Recursive Functions

Don’t do:

```maple
pow := proc( x::integer, n::nonnegint )
    if n = 0 then
        return 1;
    end if;
    return x * pow(x, n-1);
end proc:
```
Recursive Functions

Do:

```
pow := proc( x::integer, n::nonnegint )
    if n = 0 then
        return 1;
    end if;
    return x * procname(x, n-1);
end proc:
```
Loop Tricks

```maple
for i from 1 to 10 by 3 do
    ...
end do;
```

```maple
for i to 10 by 3 do
    ...
end do;
```

```maple
for i by 3 while i < 10 do
    ...
end do;
```

```maple
L := [1, 2, 3, 4];
for i in L do
    ...
end do;
```
FooBar := proc(x::integer, y::integer) local a;
    a := x/y;
    ASSERT(type(a, integer), "y must divide x");
    return a;
end proc:

> FooBar(2,3);
> kernelopts(assertlevel=1);
> FooBar(2,3);
[ Error, (in FooBar) assertion failed, assumed y|x]
Local Procedures

```
FooBar := proc ()
    rcMatMult:=proc(A::Matrix ,B::Matrix)
        return RegularChains:-MatrixTools:-MatrixMultiply(A,B,rc,R);
    end proc;
end proc:
```
Get Current Memory Usage

```maple
MemUsage := proc()
    # returns mem usage in MB
  return kernelopts(bytesalloc)/1024^2.;
end proc;

[> MemUsage();
[ 1.374748230]
```
Argument-less Procedures

```maple
FooBar := proc() 
    if nargs > 0 then 
        return args[1] + FooBar(args[2..-1]);
    end if;

    return 0;
end proc:

[> FooBar(1,2,3,4,5,6);
[ 21
```
Man Page Shortcuts

? gcd Brings you to man page for gcd.
?? gcd Jumps to gcd description.
?? gcd Jumps to gcd examples.
Debugging help

printf

Useful for printing `printf("%a", x)` will print any “algebraic object” (really the only thing you should print).
Inline if

\[
\text{IsZero} := x \rightarrow 'if'(x=0, true, false);
\]

\[
> \text{IsZero}(2);
\]

false

\[
> \text{IsZero}(0);
\]

true
Switch to symmetric mod

\[
\begin{align*}
\text{> } & \ 5 \mod 7; \\
\text{[ } & 5 \\
\text{> } & \text{'mod'} := \text{mods}; \\
\text{> } & 5 \mod 7; \\
\text{[ } & -2
\end{align*}
\]
Questions? Challenges?