Maple Lab #1 for Math 418

Note: For homework assignments: write (type!) your name at the top!

1. Basics

1.1 Help features: (a) By using ? and the command (a new window appears):

```maple
> ?sqrt
```
(b) by typing the command (or by placing the cursor on a word) and clicking on help in the toolbar:

```maple
sqrt
```

1.2 Evaluating numbers (in decimal form):

To find the value of pi to 500 decimal places, use:

```maple
> evalf(Pi, 500); 3.1415926535897932384626433832795028841971693993751058209749445923078164062862
08998628034825342117067982148086513282306647093844609550582231725359408128
4811174502841027019385211055596466294895493038196442881097566593344612847
56482337867831652712019091456485669234603486104543266482133936072602491412
73724587006606315588174881520920962829254091715364367892590360011330530548
8204665213841469519415116094330572703657595919530218611738193261179310511
85480744623799627495673518857527248912279381830119491
```

1.3 Note that Maple distinguishes between numbers and their decimal approximations:

```maple
> s2 := sqrt(2);
```

```maple
s2 := \sqrt{2} \tag{2}
```

`s2` is the exact square root of 2, whereas the following is an approximation to 50 decimal places:

```maple
> s2approx := evalf(s2, 50); s2approx := 1.4142135623730950488016887242096980785696718753769
```

1.4 Comparing numbers/expressions:

```maple
> evalb(s2^2 = 2);
```

```maple
true \tag{4}
```

```maple
> evalb(s2approx^2 = 2);
```

```maple
false \tag{5}
```

Thus, squaring `s2` is exactly equal to 2, whereas squaring `s2approx` is not.

2. Number Theory

2.1 Quotient and remainder:

```maple
> q := iquo(123, 39); r := irem(123, 39);
```

```maple
q := 3
r := 6 \tag{6}
```

Check that `q*39 + r = 123`:

```maple
> q*39 + r;
```

```maple
123 \tag{7}
```

A better way to check this is as follows:

```maple
> evalb(q*39 + r = 123);
```

```maple
true \tag{8}
```
(9) \( \text{gcd}(13843, 14351); \)  
127
(b) The extended gcd: find \( x, y \) such that \( mx + ny = \text{gcd}(m,n) \)
\( \text{igcdex}(13843, 14351, 'x', 'y'); x, y; \)  
127
28, -27
\( 13843 \cdot x + 14351 \cdot y; \)  
127

2.3 To remove the value of a variable, use the following command. (Note that if no value has been
assigned to a name (or if MAPLE doesn't recognize a command), then MAPLE returns the name (or repeats the
command).)
\( \text{unassign('x', 'y'); x, y; } \)  
\( x, y \)

2.4 The commands "ifactor" and "ifactors" both factor a given number, but give the answer in different
formats:
\( \text{ifactor}(1728); \text{ifactors}(1728); \)
\( (2)^6 (3)^3 \)
\([1, [[2, 6], [3, 3]]]\)

2.5 Prime numbers: there are several commands for finding/constructing "small" primes:
(a) The command "ithprime(n)" gives the \( n \)-th prime number:
\( \text{ithprime}(10000); \)  
104729
(b) The command "nextprime(n)" finds the first prime greater or equal to \( n \):
\( \text{nextprime}(1000); \)  
1009
(c) The command "isprime" checks whether a given number is prime:
\( \text{isprime}(1234567891); \text{isprime}(1122334455667788991); \text{isprime}(12345678901234567890987654321); \)
\text{true} \text{true} \text{false}

3. Lists
3.1 Lists and their elements:
(a) The following defines a list called "lis" (one cannot use "list" as a name):
\( \text{lis} := [5, 4, 3, 1, 2, 7]; \)  
\( \text{lis} := [5, 4, 3, 1, 2, 7] \)
(b) The 5th element of the list "lis" is obtained as follows:
\( \text{lis}[5]; \)  
2
(c) To add the element "10" at the end of the list "lis", use the following construction:
\( \text{op(lis), 10}; \)  
\( [5, 4, 3, 1, 2, 7, 10] \)
3.2 List of lists:
(a) The following is a list whose elements are other lists (and/or numbers):
> \texttt{lst := [lis, [1, 2], 5];} \\
> \texttt{lst := [[5, 4, 3, 1, 2, 7], [1, 2], 5]} \hspace{1cm} (20)

(b) The 2nd element of the list "lst" is thus the list [1,2]:
\begin{align*}
> \texttt{lst[2];} \\
> & [1, 2] \hspace{1cm} (21)
\end{align*}

(c) The double index \texttt{lst[1][2]} (= lst[1,2]) refers to the 2nd element in the list \texttt{lst[1]} (= lis):
\begin{align*}
> \texttt{lst[1, 2]; lst[1][2];} \\
> & 4 \\
& 4 \hspace{1cm} (22)
\end{align*}

For example, to get the first prime factor of 1728 = \(2^6(3^3)\) by using the program "ifactors", you would write:
\begin{align*}
> \texttt{ifactors(1728)[2, 1, 1];} \\
> & 2 \hspace{1cm} (23)
\end{align*}

because \texttt{ifactors(1728)} returns the following list of lists:
\begin{align*}
> \texttt{ifactors(1728);} \\
> & [1, [[2, 6], [3, 3]]] \hspace{1cm} (24)
\end{align*}

3.3 Constructing lists:

(a) The \texttt{sequence} command \texttt{seq(..)} is useful for constructing lists that follow a pattern:
\begin{align*}
> \texttt{seq(k^2, k = 1..10);} \\
> & [1, 4, 9, 16, 25, 36, 49, 64, 81, 100] \hspace{1cm} (25)
\end{align*}

This is the list of the first ten squares. Note that without the square brackets, we only get a \texttt{sequence}:
\begin{align*}
> \texttt{seq(k^2, k = 1..10);} \\
> & 1, 4, 9, 16, 25, 36, 49, 64, 81, 100 \hspace{1cm} (26)
\end{align*}

To construct a list of consecutive integers, you can use also the following abbreviation:
\begin{align*}
> \texttt{[10..15];} \\
> & [10, 11, 12, 13, 14, 15] \hspace{1cm} (27)
\end{align*}

(b) The "select" command allows you to pick out a sublist from a given list. For example:
\begin{align*}
> \texttt{select(isprime, [1..100]);} \\
> & [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97] \hspace{1cm} (28)
\end{align*}

gives you the table of the prime numbers less than or equal to 100.

4. Defining our own functions and procedures

4.1 Functions (simple expressions):
\begin{align*}
> f := n \rightarrow n^2 - 2 \cdot n + 1; \\
> f := n \rightarrow n^2 - 2 \cdot n + 1 \hspace{1cm} (29)
\end{align*}

This defines a function called "f". To evaluate it at \(n = 12\), write \(f(12)\):
\begin{align*}
> f(12); \\
> & 121 \hspace{1cm} (30)
\end{align*}

The following is the sequence (respectively, list) of the first 10 values of \(f(n)\):
\begin{align*}
> \texttt{seq(f(k), k = 1..10); [seq(f(k), k = 1..10)];} \\
> & 0, 1, 4, 9, 16, 25, 36, 49, 64, 81 \\
& [0, 1, 4, 9, 16, 25, 36, 49, 64, 81] \hspace{1cm} (31)
\end{align*}

4.2 Compound expressions (and/or algorithms): use "proc ... end;".

Example: Write a program "sumnos" which sums (or adds) the first \(n\) integers and returns both \(n\)
and the sum in a list, i.e., \( \text{sumnos}(n) = [n, 1 + 2 + \ldots + n] \).

\[
\text{sumnos} := \text{proc}(n) \text{ local } i, s; \newline \quad s := 0; \text{ for } i \text{ to } n \text{ do } s := s + i; \text{ od; return } ([n, s]); \text{ end;} \newline \text{sumnos} := \text{proc}(n) \text{ local } i, s; s := 0; \text{ for } i \text{ to } n \text{ do } s := s + i \text{ end do; return } [n, s] \text{ end proc} \quad (32)
\]

Note that this program uses the looping construction: \textbf{for} \ldots \textbf{do;} \ldots \textbf{(commands)} \ldots \textbf{od;}

To execute this program, say for \( n=10 \), use the program name and the value 10:

\[
\text{sumnos}(10); \quad [10, 55] \quad (33)
\]

Thus, \( 1 + 2 + \ldots + 10 = 55 \).