Maple Lab #1 for Math 211

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

1. Basics

```
> restart;    # Use this command to restart your session
```

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

```
> ?sqrt
```

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

1.2 Evaluating numbers (in decimal form):
To find the value of pi to 500 decimal places, use:

```
> evalf(Pi, 500);
```

3.14159265358979323846264338327950288419716939937510582097494459230781640628620899\
86280348253421170679821480865132823066470938446095505822317253594081284811174\
50284102701938521105559644622948954930381964428810975665933446128475648233786\
78316527120190914564856692346034861045432664821339360726024914127372458700660\
63155881748815209209628292540917153643678925903600113305305488204665213841469\
51941511609433057270365759591953092186117381932611793105118548074462379962749\
95673518857527248912279381830119491 **(1)**

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

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

$$s2 := \sqrt{2} \quad \textbf{(2)}$$

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

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

$$s2approx := 1.4142135623730950488016887242096980785696718753769 \quad \textbf{(3)}$$

1.4 Comparing numbers/expressions:

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

$$true \quad \textbf{(4)}$$

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

$$false \quad \textbf{(5)}$$

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

2. Number Theory

2.1 Fractions

```
> 49/91
```

$$\frac{7}{13} \quad \textbf{(6)}$$

Note that Maple automatically reduces the fraction to its lowest terms.

2.2 Quotient and remainder:

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

$$q := 3$$

$$r := 6 \quad (7)$$

Check that q*39 + r = 123:

```
> q·39 + r
```

$$123 \quad (8)$$

A better way to check this is as follows:

```
> evalb(q·39 + r = 123); #evalb = evaluate boolean expression
```

$$true \quad (9)$$

2.3 (a) The gcd of two numbers m, n:

```
> igcd(13843, 14351);
```

$$127 \quad (10)$$

(b) The extended gcd: find g, x, y such that mx + ny = g := gcd(m,n)

```
> igcdex(13843, 14351, 'x', 'y'); x, y;
```

$$127$$

$$28, -27 \quad (11)$$

```
> 13843·x + 14351·y;
```

$$127 \quad (12)$$

2.4 Free variables of their values. (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).)

```
> unassign('x', 'y'); x, y;
```

$$x, y \quad (13)$$

3. Lists

3.1 Lists and their elements:
List = ordered n-tuple
(a) The following defines a list called "L" (one cannot use "list" as a name):

```
> L := [5, 4, 3, 1, 2, 7];
```

$$L := [5, 4, 3, 1, 2, 7] \quad (14)$$

(b) The 5th element of the list "L" is given as follows:

```
> L[5];
```

$$2 \quad (15)$$

(c) Use "nops(..)" to determine the number of elements in a list:

```
> nops(L);
```

$$6 \quad (16)$$

3.2 List of lists:
(a) The following is a list whose elements are other lists (and/or numbers):

```
> lst := [L, [1, 2], 5];
```

$$lst := [[5, 4, 3, 1, 2, 7], [1, 2], 5] \quad (17)$$

(b) The 2nd element of the list "lst" is thus the *list* [1,2]:

```
> lst[2];
```

$$[1, 2] \quad (18)$$

(c) The double index lst[1][2] (= lst[1,2]) refers to the 2nd element in the list lst[1] (= L):

```
> lst[1, 2]; lst[1][2];
```

$$4$$

$$4 \quad \textbf{(19)}$$

3.3 The *sequence* command seq(..) is useful for constructing lists that follow a pattern:

```
> [seq(k^2, k = 1 ..10)];
```

$$[1, 4, 9, 16, 25, 36, 49, 64, 81, 100] \quad \textbf{(20)}$$

This is the list of the 1st ten squares. Note that without the square brackets, we only get a *sequence*:

```
> seq(k^2, k = 1 ..10);
```

$$1, 4, 9, 16, 25, 36, 49, 64, 81, 100 \quad \textbf{(21)}$$

4. Defining our own functions and procedures

4.1 Functions (simple expressions):

```
> f := n -> n^2 - 2·n + 1;
```

$$f := n \mapsto n^2 - 2 \cdot n + 1 \quad \textbf{(22)}$$

This defines a function called "f". To evaluate it at n = 12, write f(12):

```
> f(12);
```

$$121 \quad \textbf{(23)}$$

The following is the sequence (repectively, list) of the first 10 values of f(n):

```
> seq(f(k), k = 1 ..10); [seq(f(k), k = 1 ..10)];
```

$$0, 1, 4, 9, 16, 25, 36, 49, 64, 81$$

$$[0, 1, 4, 9, 16, 25, 36, 49, 64, 81] \quad \textbf{(24)}$$

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., sumnos(n) = [n, 1 + 2 + ... + n].

```
> sumnos := proc(n) local i, s;
  s := 0; for i from 1 to n do; s := s + i; od; return([n, s]); end;
```

$$sumnos := \textbf{proc}(n)\ \textbf{local}\ i, s;\ s := 0;\ \textbf{for}\ i\ \textbf{to}\ n\ \textbf{do}\ s := s + i\ \textbf{end do};\ \textbf{return}\ [n, s]\ \textbf{end proc} \quad \textbf{(25)}$$

Note that this program uses the looping construction: **for ... do; ...** (commands) ... **od;**

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

```
> sumnos(10);
```

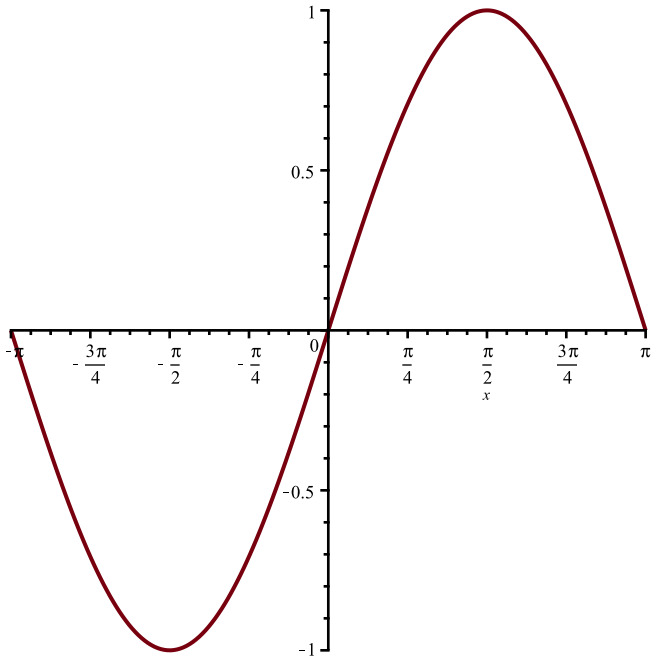
$$[10, 55] \quad \textbf{(26)}$$

Thus, 1 + 2 + ... + 10 = 55.

5. Plots/Graphics

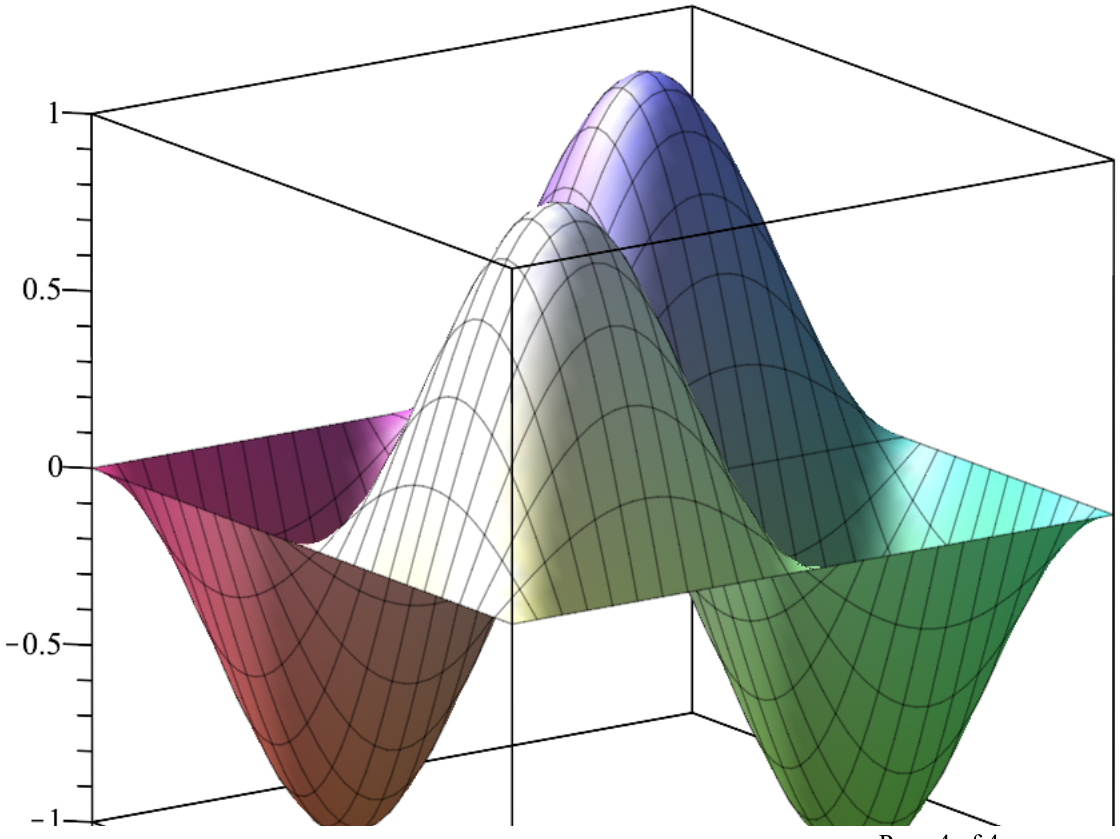
5.1 Drawing 2-dimensional plots:

```
> plot(sin(x), x = -Pi ..Pi);
```



5.2 Drawing 3-dimensional plots: use "plot3d(..)".
Note that you can rotate the 3-D plot with your cursor. Try it!

```
> plot3d(sin(x)·sin(y), x=-Pi..Pi, y=-Pi..Pi);
```




```
>
```