Maple Lab #1 for Math 418

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

1. Basics

> *restart*;

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);

3.14159265358979323846264338327950288419716939937510582097494459230781640628620\
08998628034825342117067982148086513282306647093844609550582231725359408128\
48111745028410270193852110555964462294895493038196442881097566593344612847\
56482337867831652712019091456485669234603486104543266482133936072602491412\
73724587006606315588174881520920962829254091715364367892590360011330530548\
82046652138414695194151160943305727036575959195309218611738193261179310511\
854807446237996274956735188575272489122793818301194912 **(1)**

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

> $s2 := \mathrm{sqrt}(2)$;

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

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

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

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

1.4 Comparing numbers/expressions:

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

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

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

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

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

2. Number Theory

2.1 Quotient and remainder:

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

$$q := 3$$
$$r := 6 \qquad \textbf{(6)}$$

Check that q*39 + r = 123:

> $q \cdot 39 + r$;

$$123 \qquad \textbf{(7)}$$

A better way to check this is as follows:

> $evalb(q \cdot 39 + r = 123)$;

$$true \qquad \textbf{(8)}$$

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

> $igcd(13843, 14351);$

$$127 \tag{9}$$

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

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

$$127$$
$$28, -27 \tag{10}$$

> $13843 \cdot x + 14351 \cdot y;$

$$127 \tag{11}$$

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).)

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

$$x, y \tag{12}$$

2.4 The commands "ifactor" and "ifactors" both factor a given number, but give the answer in different formats:

> $ifactor(1728); ifactors(1728);$

$$(2)^6 (3)^3$$
$$[1, [[2, 6], [3, 3]]] \tag{13}$$

2.5 Prime numbers: there are several commands for finding/constructing "small" primes:
(a) The command "ithprime(n)" gives the n-th prime number:

> $ithprime(10000);$

$$104729 \tag{14}$$

(b) The command "nextprime(n)" finds the first prime greater or equal to n:

> $nextprime(1000);$

$$1009 \tag{15}$$

(c) The command "isprime" checks whether a given number is prime:

> $isprime(1234567891); isprime(1122334455667788991);$
> $isprime(1234567890123456789098765 4321);$

$$true$$
$$true$$
$$false \tag{16}$$

3. Lists

3.1 Lists and their elements:
(a) The following defines a list called "lis" (one cannot use "list" as a name):

> $lis := [5, 4, 3, 1, 2, 7];$

$$lis := [5, 4, 3, 1, 2, 7] \tag{17}$$

(b) The 5th element of the list "lis" is obtained as follows:

> $lis[5];$

$$2 \tag{18}$$

(c) To add the element "10" at the end of the list "lis", use the following construction:

> $[op(lis), 10];$

$$[5, 4, 3, 1, 2, 7, 10] \tag{19}$$

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

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

$$lst := [[5, 4, 3, 1, 2, 7], [1, 2], 5] \quad (20)$$

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

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

$$[1, 2] \quad (21)$$

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

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

$$4$$
$$4 \quad (22)$$

For example, to get the first prime factor of 1728 = (2^6)(3^3) by using the program "ifactors", you would write:

```
> ifactors(1728)[2, 1, 1];
```

$$2 \quad (23)$$

because ifactors(1728) returns the following list of lists:

```
> ifactors(1728);
```

$$[1, [[2, 6], [3, 3]]] \quad (24)$$

3.3 Constructing lists:

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

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

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

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

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

$$1, 4, 9, 16, 25, 36, 49, 64, 81, 100 \quad (26)$$

To construct a list of consecutive integers, you can use also the following abbreviation:

```
> [$10..15];
```

$$[10, 11, 12, 13, 14, 15] \quad (27)$$

(b) The "select" command allows you to pick out a sublist from a given list. For example:

```
> select(isprime, [$1..100]);
```

$$[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97] \quad (28)$$

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):

```
> f := n -> n^2 - 2·n + 1;
```

$$f := n \rightarrow n^2 - 2\,n + 1 \quad (29)$$

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

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

$$121 \quad (30)$$

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)];
```

$$0, 1, 4, 9, 16, 25, 36, 49, 64, 81$$
$$[0, 1, 4, 9, 16, 25, 36, 49, 64, 81] \quad (31)$$

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 to n do; s := s + i; od; return([n, s]); end;
```

$sumnos := \mathbf{proc}(n)\ \mathbf{local}\ i, s;\ s := 0;\ \mathbf{for}\ i\ \mathbf{to}\ n\ \mathbf{do}\ s := s + i\ \mathbf{end\ do};\ \mathbf{return}\ [n, s]\ \mathbf{end\ proc}$ **(32)**

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);
```

$$[10, 55] \quad \textbf{(33)}$$

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