**Chapter 3 Short Summary**

Remark: the order of function arguments matter.

\* *sum()* and *product()*

**> ** = 

**> **= 

These functions are intended for symbolic computations.

\* *add()* and *mul()*



These functions are intended for adding or multiplying explicit finite sequences or expressions.

e.g.  = .

e.g. **> **

.

These do not accept symbolic ranges and cannot remain unevaluated (i.e. computational functions). They are much more efficient than sum() and product().

(Remark: add() and mul() also accept data structures other than ranges, and instead of setting the control variable to each integer in a range, set it to each top-level operand of a data structure. Look at lecture notes for examples.)

\* *eval()*

*eval()* evaluates an expression at particular values (for any number of variables), e.g.

 = 

 = .

-> You can evaluate functions using **** = ****, or **** from the expressions palette.

e.g.

**** = ****.

-> For two variables: e.g.

**** = ****,

**** = ****,

**** = ****.

Remark: Evaluating sequentially (e.g.  = ) is the same as evaluating simultaneously, unless the substituted values involve the original variables.

\* *limit()*

-> Available in Calculus palette or use *limit(f, x=a)*.

-> Right and Left-hand limits: use *limit(f, x=a, right/left).*

Or, type a +/- next to the template. e.g.

Left-hand:  = 

Right-hand:  = .

-> Note that *limit()* cannot remain symbolic.

\* *taylor()* and *series()*

-> *taylor(f(x), x)* computes a Maclaurin polynomial of f(x) with respect to x, of degree 5 by default.

e.g.

 **> **



(where O(x7) is the big-O notation).

-> Maclaurin series can also be computed using functions encountered before. i.e.

, or, 

*-> series()* is a more general version of *taylor()*. See Maple help for more details on *series()* and *taylor()*.

\* *Inert* functions and *value()*

*Inert* functions do not perform evaluation, but may be displayed specially and/or recognized as data by other functions. E.g.

**** = 

**** = ****

**** = ****

**** = ****

**** = ****

**** = ****

The function *value()* converts an inert function into an active function and evaluates it, e.g.

 **> **= 

Remark: Standard inert functions normally have the first letter of their name capitalized.

Remark: The main structure is shown in grey, as a reminder that the function is inert.

Remark: Inert functions can be evaluated in special ways by other functions. See lecture notes.

\* *Subscripts*

-> Enter an active subscript by using the  template in the Expression palette.

-> In both 1-D and 2-D input modes, you can also use *L*[*i*], which is the same as *Li*.

-> In Maple 17 and later versions, you can also enter an active subscript by typing (CTRL & SHIFT & -).

\* *evalf()*

-> *evalf()* performs numerical approximation of its argument (to 10 significant figures by default), e.g.

**> **



-> change the number of sig. figures using a subscript

e.g.

 = or  = .

(Remark: evalf() will evaluate any components of an expression that have numerical values and leave the rest of the expression intact.)

(\* *Approximating definite integrals*)

A definite integral with the integration variable as its only variable, should always have a numeric approximation.

e.g.

**> **



**> **



In situations like this, Maple may spend a long time trying and failing to evaluate the integral exactly. If you know that you want a numerical approximation, it may be significantly faster to use an inert integral, like this:

**> **



or equivalently to give the function *int* a third argument of *numeric*, like this:

**> **



To specify the precision, use a fourth argument like this:

**> **



(Remark: In non-trivial cases where Maple can evaluate a definite integral exactly, you may get not only a faster but also a more accurate or more appropriate numerical result by using only numerical integration. Look at lecture notes for example.)

\* *simplify()*, *expand()*, *factor()*

-> Maple only does the most basic simplifying automatically. The main tool for simplifying further is *simplify()*.

-> For polynomials, you can use *expand()* or *factor()* if *simplify()* does not do what you want. (Note that factor() will only work with polynomails)

e.g.

**> **



**> **



**> **



-> You can have more control over the simplification options by providing second arguments such as ‘symbolic’. See maple help and lecture notes for more details.

\* Prime functions: *isprime(), nextprime(), prevprime(), ithprime(), ifactor()*

-> *isprime()* tells you if a number is prime. E.g.  ****= , ****= 

-> *nextprime()* and *prevprime()* require their arguments to be integers and return respectively the next largest or next smallest prime.

-> *ithprime()* requires its argument to be a positive integer *i* and returns the ith prime.

-> *ifactor()* finds the prime factorization of an integer, e.g.

**** = 

(Applying the function expand() to the result gives the original integer back).