Before you start, in the *text mode*, enter

```
Your name
Date
```

Then, switch to *math mode*, enter

```
> with(student):with(plots):
```

to load the student and plots packages.

The `for` loop

We would like to find a way to repeat a command several times in the most efficient manner. For example, suppose you wanted to evaluate 1^2+1, 2^2+1, 3^2+3, $\ldots$, 8^2+8, you could write the following commands

```
> 1^2+1;
> 2^2+2;
  ...
> 8^2+8;
```

Note that this is very time consuming. However, these numbers all appear in the form i^2+i, where i runs from 1 to 8. Hence, we would like to write a general statement that can be repeated. Note that the general statement would be

```
> i^2+i;
```

where the only problem is that *Maple* needs to know what values of i we are trying to deal with. This can be achieved easily with a **for**-loop. The syntax for a for-loop has the structure **for ... from ... by ... to ... do**, in our case, we will write

```
> for i from 1 by 1 to 8 do
  i^2+i;
  od;
```

In this loop, the commands *in between* `do ... od` are executed repeatedly as i increases from 1 to 8, every time by 1. The loop starts with $i=1$, executes i^2+i, then i is increase by 1 to 2, and executes i^2+i again. This process goes on until $i=8$.

Notice that there is not a semi-colon after the first line, and that **the line between `do` and `od` constitutes the do-loop code**.

Note that we can enter multiple lines of Maple code without execution by pressing <Shift> + <Enter> instead of just the <Enter> key. At the end of the loop, press <Enter> to execute the code.

For example, suppose you wanted to evaluate $f(x) = \sin(x)\cos(x)$ at the first ten multiples of $\pi/6$. You could write the following ten commands:

```
> f:=sin(x)*cos(x);
> subs(x=Pi/6,f);
> subs(x=2*Pi/6,f);
  ...
> subs(x=10*Pi/6,f);
```

Instead, notice that the commands have a repeating pattern of

```
> subs(x=i*Pi/6,f);
```

where i is an integer index that ranges from 1 to 10. Hence a for-loop can be written as

```
> for i from 1 to 10 do
  subs(x=i*Pi/6,f);
  od;
```

Note that we can take away the part `by 1` when the increase in i is just 1 every time.

Exercises:

(1) Consider the quadratic function $f(x) = 2x^3 - 4x + \frac{5}{x^2}$.

- **(a)** Define f as a **function** in *Maple*.
- **(b)** Write a **for** loop to evaluate the values of $f(1), f(2), \ldots, f(10)$.
- **(c)** Write a **for** loop to evaluate the values of $f(1), f(2), f(4), f(8), \ldots, f(2^{16})$.
- **(d)** Write a **for** loop to evaluate the values of $f(4), f(7), f(10), \ldots, f(28)$.

(2) Use a for-loop to verify the summation formula $\displaystyle\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$. Begin by letting $n = 10$ and consider the following code:

```
> total:=0;
> for i from 1 to 10 do
  total:=total+i;
  od;
```

Step through the do-loop and keep track of the value for `total`. Notice that this code finds the total sum by adding the next term to the previous sum at each step. Also notice that `total:=0` is assigned before the start of the loop (why?). Test your understanding of this code by implementing a do-loop that shows

$$\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}.$$

Test your code against the formula for $n = 20$.

The `if...then...fi` Command

Suppose you wanted to stop executing a for-loop before i has reached its final value. For example, suppose you let $n = 20$ but want to find out how many terms add up to 200, for the terms i^2 in 2.3. After you calculate `total`, you could check if `total` is 200 or more. If so, we want to `break` the loop; if not, we want the loop to `continue`. Here is how you can write this conditional `if` statement in *Maple*:

```
> total:=0;
> for i from 1 to 20 do
    i;total:=total+i;
    if total > 200 then break;
      else continue;
    fi;
  od;
```

Exercises:

(3) Find the number of terms needed so that

$$\sum_{i=1}^{n} i^2 > 500$$

(4) The function `tau(n)` in the `numtheory` package counts the number of positive divisors of of a number n, while the function `divisors(n)` list all the divisors. Start by loading the package by typing in

```
> with(numtheory):
```

(a) Check the function `tau(n)` and `divisors(n)` on the numbers 60 and 720.

(b) Use a for loop to find the smallest integer such that `tau(n)`> 6. Display the divisors for this number.