subs

The subs command is used to make substitutions of one expression (variable, number...) for another within some other (presumably more complicated) Maple expression. In other words, subs can be used for "plugging-in". The most important thing to remember about subs is that it merely reports what the result of making a substitution would be -- it does NOT change the value of any variable (unless the result of subs is assigned to a variable).

A few simple examples will give you the idea: Suppose

```
> y:=x+3;
```

$$y := x + 3$$

Now you know that if you substitute 2 for x, y would be 5 -- to see this in Maple, type:

```
> subs(x=2,y);
```

$$5$$

Most importantly, after that statement it is still the case that:

```
> x;
```

$$x$$

i.e., x has no value (because it didn't before), and

```
> y;
```

$$x + 3$$

y is still what is was before.

The syntax of subs is the standard Maple:

```
subs( what , how );
```

syntax -- the "what" says what substitution (or set of substitutions) is to be made. The "how" is what expression to make the substitutions in. There is one deviation from standard Maple syntax -- if you have more than one substitution to make, it is

not necessary (although it is permitted) to enclose the list of substitutions in braces. Thus, the following is a valid statement:

```
> z:=x+sin(2*h):
> subs(x=3*u+1,h=4*Pi*q,z);
```

$$3\,u + 1 + \sin(8\,\pi\,q)$$

Notice that the things being substituted can be quite complicated.

The thing being substituted for can also be complicated:

```
> subs(sin(2*h)=4*w^2-1,z);
```

$$x + 4\,w^2 - 1$$

What to watch out for: Aside from syntax errors, there are few things that can go haywire when using `subs` - two stand out:

1. In certain situations, `subs` will not be able to find the expression you are substituing for if it is complicated (as in the last example) -- this has to do with the way Maple stores expressions internally. It is sometimes possible to "re-word" your substitution request so that Maple "gets it"- or else to break the task into several manageable ones.

2. You need not fear direct "circular substitutions" - for instance, using z from above:

```
> z;
```

$$x + \sin(2\,h)$$

attempt the substitution:

```
> subs(x=2*x,z);
```

$$2\,x + \sin(2\,h)$$

On the other hand, long chains of self-referential substitutions may produce unpredictable results. Could you have predicted the result of the following?

```
> subs(h=x,x=z,z);
```

$$x + \sin(2\,h) + \sin(2\,x + 2\,\sin(2\,h))$$

The following is different because substitutions in braces are made *simultaneously* instead of sequentially left-to-right:

```
> subs({h=x,x=z},z);
```

$$x + \sin(2\,h) + \sin(2\,x)$$

If things get too self-referential, Maple may generate a "stack overflow" message. The moral is that self-referential substitutions should be generally avoided, or done at most one at a time.

Why `subs` is useful: More often than not, it is better to use `subs` rather than to make assignments to the variables in an expression. This is because `subs` reports the result of making a substitution without actually affecting the values of any variables. This leaves the definitions of the variables intact for later use.

For example, if

```
> y:=3*x^2+5*x+2;
```

$$y := 3\,x^2 + 5\,x + 2$$

and you want to compute the difference quotient (y(x+h)-y(x))/h, then you can write

```
> dq:=(subs(x=x+h,y)-y)/h;
```

$$dq := \frac{3\,(x+h)^2 + 5\,h - 3\,x^2}{h}$$

We cannot set:

```
> x:=x+h;
Warning, recursive definition of name
```

$$x := x + h$$

since, as Maple is warning us, x now is defined (circularly) in terms of itself. Subsequently, any attempt to evaluate y will result in:

```
> y;
Error, too many levels of recursion
```

You could try just setting

```
> x:=4;
```

$$x := 4$$

```
> y;
```

$$70$$

That's fine, but now y is a number -- we can't plug any other values of x into it. So, `subs` is a better way to do plugging-in.

A final note: The `subs` command is not what to use to do changes of variable in integrals (because it doesn't know about the dx part) - there is another Maple command, `changevar` in the `student` library, that is used for this purpose.