```
>> mynum = mynum + 1
mynum =
         7
```

In the first assignment statement, the value 3 is assigned to the variable *mynum*. In the next assignment statement, *mynum* is changed to have the value of the expression 4 + 2, or 6. In the third assignment statement, *mynum* is changed again, to the result of the expression mynum + 1. Since at that time *mynum* had the value 6, the value of the expression was 6 + 1, or 7.

At that point, if the expression mynum + 3 is entered, the default variable *ans* is used since the result of this expression is not assigned to a variable. Thus, the value of *ans* becomes 10 but *mynum* is unchanged (it is still 7). Note that just typing the name of a variable will display its value.

```
>> mynum + 3
ans =
      10
>> mynum
mynum =
       7
```

1.2.1 Initializing, Incrementing, and Decrementing

Frequently, values of variables change. Putting the first or initial value in a variable is called ***initializing*** the variable.

Adding to a variable is called ***incrementing***. For example, the statement

```
mynum = mynum + 1
```

increments the variable *mynum* by 1.

QUICK QUESTION!

How can 1 be subtracted from the value of a variable called *num*?

Answer:

```
num = num – 1;
```

This is called ***decrementing*** the variable.

1.2.2 Variable Names

Variable names are an example of ***identifier names***. We will see other examples of identifier names, such as filenames, in future chapters. The rules for identifier names are:

- The name must begin with a letter of the alphabet. After that, the name can contain letters, digits, and the underscore character (e.g., value_1), but it cannot have a space.
- There is a limit to the length of the name; the built-in function **namelengthmax** tells how many characters this is.
- MATLAB is case-sensitive. That means that there is a difference between upper- and lowercase letters. So, variables called *mynum*, *MYNUM*, and *Mynum* are all different.
- There are certain words called ***reserved words*** that cannot be used as variable names.
- Names of built-in functions can, but should not, be used as variable names.

Additionally, variable names should always be ***mnemonic***, which means they should make some sense. For example, if the variable is storing the radius of a circle, a name such as "radius" would make sense; "x" probably wouldn't.

The Workspace Window shows the variables that have been created in the current Command Window and their values.

The following commands relate to variables:

- **who** shows variables that have been defined in this Command Window (this just shows the names of the variables)
- **whos** shows variables that have been defined in this Command Window (this shows more information on the variables, similar to what is in the Workspace Window)
- **clear** clears out all variables so they no longer exist
- **clear** *variablename* clears out a particular variable

If nothing appears when **who** or **whos** is entered, that means there aren't any variables! For example, in the beginning of a MATLAB session, variables could be created and then selectively cleared (remember that the semicolon suppresses output):

```
>> who
>> mynum = 3;
>> mynum + 5;

>> who
Your variables are:
ans  mynum

>> clear mynum
```

```
>> who
Your variables are:
ans
```

1.3 EXPRESSIONS

Expressions can be created using values, variables that have already been created, operators, built-in functions, and parentheses. For numbers, these can include operators such as multiplication, and functions such as trigonometric functions. An example of such an expression would be:

```
>> 2 * sin(1.4)
ans =
     1.9709
```

1.3.1 The Format Function and Ellipsis

The ***default*** in MATLAB is to display numbers that have decimal places with four decimal places, as already shown. The **format** command can be used to specify the output format of expressions. There are many options, including making the format **short** (the default) or **long**. For example, changing the format to **long** will result in 15 decimal places. This will remain in effect until the format is changed back to **short**, as demonstrated with an expression and with the built-in value for **pi**.

```
>> format long
>> 2 * sin(1.4)
ans =
     1.970899459976920

>> pi
ans =
     3.141592653589793

>> format short
>> 2 * sin(1.4)
ans =
     1.9709

>> pi
ans =
     3.1416
```

The **format** command can also be used to control the spacing between the MATLAB command or expression and the result; it can be either **loose** (the default) or **compact**.

```
>> format loose
>> 2^7

ans =

     128

>> format compact
>> 2^7
ans =
     128
```

Especially long expressions can be continued on the next line by typing three (or more) periods, which is the continuation operator, or the **ellipsis**. For example,

```
>> 3 + 55 - 62 + 4 - 5 ...
   + 22 - 1

ans =
     16
```

1.3.2 Operators

There are in general two kinds of operators: ***unary*** operators, which operate on a single value or ***operand***; and ***binary*** operators, which operate on two values or operands. The symbol "–", for example, is both the unary operator for negation and the binary operator for subtraction.

Here are some of the common operators that can be used with numeric expressions:

+	addition
–	negation, subtraction
*	multiplication
/	division (divided by e.g. 10/5 is 2)
\	division (divided into e.g. 5\10 is 2)
^	exponentiation (e.g., 5^2 is 25)

1.3.2.1 Operator Precedence Rules

Some operators have ***precedence*** over others. For example, in the expression 4 + 5 * 3, the multiplication takes precedence over the addition, so first 5 is multiplied by 3, then 4 is added to the result. Using parentheses can change the precedence in an expression:

```
>> 4 + 5 * 3
ans =
    19

>> (4 + 5) * 3
ans =
     27
```