✓ Add the pipe | symbol to the right of the method block declaration followed by the two variable names separated by commas, then test the program verify it still functions properly.

```
PUB ShiftLedsLeft | pattern, divide
```

Aside from the fact that the `pattern` and `divide` variables are now local, meaning other methods in the object could not use them; since our object has just one method this is of no consequence here. There is one other difference. When we used the `VAR` block syntax, we had the option of defining our global variables as byte, word, or long in size. However, local variables are automatically defined as longs and there is no option for byte or word size local variables.

Timekeeping Applications

For clock and timekeeping applications, it's important to eliminate all possible errors, except for the accuracy of the crystal oscillator. Take a look at the two objects that perform timekeeping. Assuming you have a very accurate crystal, the program on the left has a serious problem! The problem is that each time the loop is repeated, the clock ticks elapsed during the execution of the commands in the loop are not accounted for, and this unknown delay accumulates along with **`clkfreq + cnt`**. So, the number of seconds the `seconds` variable will be off by will grow each day and will be significantly more than just the error introduced by the crystal's rated +/- PPM.

The program on the right solves this problem with two additional variables: `T` and `dT`. A time increment is set with `dT := clkfreq` which makes `dT` equal to one second with the precision of the crystal. A particular starting time is marked with `T := cnt`. Then, inside the loop, it recalculates the next **`cnt`** value that **`waitcnt`** will have to wait for with `T += dT`. This use of the add assignment operator **`+=`** allows us to create a precise offset from original marked value of `T`. With this system, each new target value for **`waitcnt`** is exactly 1 second's worth of clock ticks from the previous. It no longer matters how many tasks get performed between **`waitcnt`** command executions, the program on the right will never lose any clock ticks and maintain a constant 1 s time base that's as good as the signal that the Propeller chip is getting from the external crystal oscillator.

```
''File: TimekeepingBad.spin

CON

    _xinfreq = 5_000_000
    _clkmode = xtal1 + pll1x

VAR

    long seconds

PUB BadTimeCount

    dira[4]~~

    repeat
       waitcnt(clkfreq + cnt)
       seconds ++
       ! outa[4]
```

```
''File: TimekeepingGood.spin

CON

    _xinfreq = 5_000_000
    _clkmode = xtal1 + pll1x

VAR

    long seconds, dT, T

PUB GoodTimeCount

    dira[9..4]~~

    dT := clkfreq
    T  := cnt

    repeat
       T += dT
       waitcnt(T)
       seconds ++
       outa[9..4] := seconds
```

✓ Try running both objects. Without an oscilloscope, there should be no noticeable difference.