This exercise is meant to help you get used to the Occam 1 programming language and the Browser Occam website. For a full reference of Occam, see this page (tba).

**Exercise 1.** Take a look at the following code.

```
SEQ
    SEQ i = [0 FOR 31]
        GRAPHICS[i][0] ! 3
    SEQ i = [0 FOR 31]
        GRAPHICS[i][0] ! 7
```

It will be connected to a screen where 3 is the code for red and 7 is the code for blue, so `GRAPHICS[i][0] ! 3` will draw a red pixel to the screen at position (i,0).

What do you think the code will do? Paste it into the website's text box, run it and see if you were correct.

**Exercise 2.** As you may have noticed, the `SEQ` keyword doesn't seem to do anything on its own. Unlike, for example, a `WHILE` statement which runs the code in the body until the condition becomes false, a `SEQ` statement simply runs each piece of code in the body, one after the other, like in any ordinary programming language.

However, in `occam`, executing code in order is not the only option. We can also execute code *concurrently* with the `PAR` keyword instead. Each piece of code becomes a separate *process*.

Take a look at this version of the code, which has the outer `SEQ` replaced with `PAR`. How do you think it will behave?

```
PAR
    SEQ i = [0 FOR 31]
        GRAPHICS[i][0] ! 3
    SEQ i = [0 FOR 31]
        GRAPHICS[i][0] ! 7
```

Try running it in the browser. Were you correct?

**Exercise 3.** `occam` has a feature called *blocking channels*. You can send a value on a channel with `!`, e.g.

```
myChannel ! 1
```

You can also receive from the channel into a variable with `?`:

```
myChannel ? x
```

However, when using `!`, the process will have to wait until someone receives its value before moving on. And when using `?`, the process will have to wait until someone sends it a value before it can move on. We say that the process *blocks* until the channel communication is successful. (Note that we do not block when

writing to `GRAPHICS` with `!`. Think of the screen as being constantly ready to receive messages.)

So, the following code will block forever after drawing a red dot, because we will never move past the sending line `myChannel ! 1`:

```
SEQ
    CHAN OF INT myChannel:
    INT x:
    SEQ
        SEQ
            GRAPHICS[0][0] ! 3
            myChannel ! 1
        SEQ
            myChannel ? x
            GRAPHICS[0][0] ! 7
```

However, if you were to change the second `SEQ` to a `PAR`, then the processes sending and receiving on the channel would become concurrent. Whichever used the channel first, would wait until the second one used the channel too, and then they would both succeed simultaneously. After that, the second process would be able to draw a blue dot. Try it for yourself.

How would you use a channel to modify the code from Exercise 2, so that it behaves the same as the code from Exercise 1? Paste your code into a text file.

**Exercise 4. (Bonus: Making it rain)** The following code colours the screen blue from top to bottom, with each column of pixels coloured by a different process.

```
SEQ
    [32]INT verticalPosition:
    SEQ i = [0 FOR 31]
        verticalPosition[i] := 0
    PAR i = [0 FOR 31]
        WHILE verticalPosition[i] < 32
            SEQ
                GRAPHICS[i][verticalPosition[i]] ! 7
                verticalPosition[i] := verticalPosition[i]+1
```

You can receive number key presses, e.g. to a variable `x` with `KEYBOARD ? x`. Here is a piece of code which advances a red dot from left to right across the screen as you press the `1` key.

```
SEQ
    INT me:
    me := 0
    INT keypress:
    WHILE TRUE
        SEQ
```

```
GRAPHICS[me][0] ! 3
KEYBOARD ? keypress
GRAPHICS[me][0] ! 0
me := me+1
```

Try combining those two pieces of code, so that the red dot sets off the blue process in each column that it visits. You will need to use an array of channels, for example declaring them as `[32]CHAN OF INT visited:` Paste your code into a text file.