Laboratory 3

Objective

In this week’s laboratory we will investigate the use of variables in computer software. We will create and manipulate the three main types (numbers, text and boolean) of variables. We will look at the main functions provided by Scribble to provide input and output.

The aim of the laboratory is to help you:

1. learn how to create and use variables in Scribble
2. understand the three types of variables in Scribble and the different types of operators they can be used with
3. learn to perform basic calculations using variables with Scribble
4. learn to perform basic string manipulation using variables with Scribble
5. consolidate your understanding of the link between a flowchart and program code
6. learn how to use the basic input (ask) and output (say) functions of Scribble

Required resources

To perform this laboratory at Monash:

* You will need a Windows-based PC in a Monash student laboratory
* Your authcate credentials to login to the computer and to the unit-web site on Moodle

To perform this laboratory at home on your own computer:

* A working copy of Scribble
* Your authcate credentials to login to the computer and to the unit-web site on Moodle

Tasks (on-campus students)

1. Making a variable and giving it a value. Variables are an important part of any software application. It’s pretty easy to create a variable in Scribble. Start a new Scribble application. For the moment we are just going to make some simple scripts, that we will run by double
clicking, in the default sprite (Sprite1). To create a variable just go to the Variables code block group in the scripting toolbox. The first item there is a grey button labelled “Make a variable” Click that button, a dialog appears asking you to specify couple of options for the variable that is being created.

![Dialog for creating a variable]

You should choose meaningful names for your variables. That will make your code easier to understand - esp. by others, and easier to debug when something is wrong. With that advice in mind, create a variable - called \( x \). (Despite what I just wrote, it is common for variables that are used for simple counting or housekeeping purposes to be given simple names like \( x \), \( y \) and \( z \). It's not great practice, but it's common.)

Notice the other option available at the time you create a variable in Scribble allows for you to specify the scope of the variable. More on this in this week's lecture and also later in the unit: variable scope is an important concept for you to understand. It's a powerful tool to use to make sure the code you write in maintainable. In general, making variables “global” makes your work as a programmer easier in the short term and harder in the long term. It's considered good practice to restrict the scope of variables as much as possible. Ignoring that, simply make the scope of this variable \( x \) global - that's available for all sprites. Click the OK button and the variable is created. You should immediately notice a couple of changes to the screen. The variable \( x \) should be added as a drag-able item in the Variables code block group. You should also notice that the variable appears in the stage area. See the little check box next to the variable where it appears in the Variables code block group. Click that check box to turn it off, and you’ll see the display of the variable in the stage area vanishes. Click it again and it reappears. That’s a neat device for showing the values of variables while your code is running, and then once you have made sure your code is working correctly you can remove them from the main display making it look cleaner.
2. **A look back at the lilac chaser: generalising code with variables.** Do you have a copy of the lilac chaser application you created in laboratory 1? If you do open that now. If not, download a copy from the unit web site. Add a variable to that application, and name it `waittime`. Make its scope global, so that it is available in all sprites. In the script that belongs to the canvas - the one that reacts to the click of a green flag and starts the process of the dots flashing by broadcasting a message named “show dot 1” add a set code block that sets the value of the variable `waittime` to 0.1.

Now, in each of the dots, find the `wait` command in the each of the scripts. Drag the variable `waittime` (from the `Variables` code block group) into the body of the `wait` command - replacing the value 0.1 that you typed there with the variable `waittime`. 
1. After the variable is created it appears here. It can be dragged into a script.

2. Drag the variable \textit{waittime} into the editable part of the \texttt{wait} command (where the time to wait in seconds can be typed).

Once you have done that, do it for all the dots (sprites) in your application. As \textit{waittime} has a value of 0.1, that value will be the time (in seconds) that is used in executing the \texttt{wait} command so the application should run just the same as before after you make all the changes. Verify that this is the case.

? Find a partner in your laboratory and discuss why you were asked to do that. I was more work. Are there any benefits? This section was headed ... generalising your code ... what did that mean?

\textbf{2.1.} It’s now possible by changing the value of \textit{waittime} in the script that belongs to the canvas (where it’s value is initialised) to give it a value other than \texttt{.1} and for all the flashing of the dots in the circle to speed up or slow down. try that with a few different values, try \texttt{0.01} and \texttt{1.0}. After you have done that, set the value back to \texttt{.1}. Now use the click box next to the variable name (on display in the group of code blocks) to turn on the display of the variable in the stage area. Click on the variable there. You’ll notice that is cycles through thee different forms of display. The default which shows the variable name and it’s value, a slider that has an interactive slider bar under the number and a final display that just shows the value and not the name of the variable. Click to make the slider bar form appear. Then right click on the slider bar.

1. Double click on the variable display on the stage and the “type” of display changes

2. First time is shows a slider. This is interactive, manipulating the slider changes the variable’s value.

3. Second time it removes the variable’s name and just shops the value. Double click again and it goes back to 1.

4. When the slider view is selected, right clicking on the item displays a pop-up menu. From this menu you can set the min and max value of the slider.
3. Basic input and output with Scribble. The say and ask commands. We are about to make one of the dumbest programs you could imagine - it’s standard fare in introductory courses on programming - so please forgive me. It’s a program that asks an end user for their name and then just prints (using the say command) it back to them. We are going to create a version of it in Scribble to illustrate the use of the ask and say commands, the answer variable and the join operator. All important and simple command blocks that in a shirt time you will be able to use without thinking. Start a new project, and in the default sprite (Sprite1) create the following script.

Begin by making a variable called name. Then start a script using the ask command block from the command block group. Connect a set command to that, make the focus of the set command the variable name. Go back to the command block group, you’ll notice there is a built-in variable there named answer. This built-in variable stores the value of whatever input the user has provided when an ask code block runs. Drag that answer variable into the editable part of the set command (the result when it runs will put the user input into the name variable). Now, snap a say command to the bottom of the script (from the . The say command in Scribble is used for simple display of results. The “thing” that is output can be text or the result of the evaluation of some expression. The output we want for this script is going to combine two pieces of text - the text “Your name is ” and the value stored in the variable name. To do that we will use a join operator. Look for that in the code block group. Drag the join operator into the say command. Then type the text “Your name is ” into the first part of is, then drag the variable name into the second part. After that test your script by double clicking on it. When prompted by the sprite, type in your name and the script should then display the result.
The script uses all the commands mentioned earlier: `ask`, `say`, `set`, the join operator and the built-in variable `answer`. None of that was hard, but it is really important that you understand how those commands work as you will use them a lot. This is the result of running the script when I typed in my name (Peter). Your’s should work in a similar manner.

Even though that’s a simple script, let’s explore it a bit more. Let’s consider for the moment the `join` operator. It joins two items of text together. The text items might be something typed in to the operator directly (like the text “Your name is ”, or a value in a variable, or the result of the evaluation of an expression. One of the things that can be joined - is the result of another join. This is a technique (often used in programming) called nesting. By nesting join statements you can make text output that is made up of any number of items. Let’s change our script to include nested `join` operators in the `say` command.

1. This `say` command has been completed by using two `join` statements. One dropped inside the other. This is called nesting.

2. This is the output of that script if the end-user has typed “Peter” in response to the prompt “What’s your name?”

Nesting commands like that is very useful. It allows simple program elements to create much provide complex functionality. It can, however, make program code hard to follow. One way of helping keep the scripts you write readable (and as a result easier to debug and later to maintain) is to break up expressions (not just ones that are nested) and store intermediate results in variables. Here is an example of a script - not that two joins nested is hard to read - that does the same work as the previous script but its separates the two joins. The `say` command simply outputs the value stored in the variable `greeting`.
make your script like that. You’ll need to create two variables: salutation and greeting, and add some more set code blocks to your script. Verify that your script works as before.

Another point not to be lost is the flexibility of the say code block. It can output text that has been typed in, the value of a variable, or it can evaluate an expression and output the result. In a moment we will create some boolean expressions and use the say command to show the results.

4. **String manipulation.** Build on the script you have just made. Make the script ask the user for their firstname, their middle name and their last name. Then output the sentence “Your name is Name”. Make Name be the first initial, their second initial a space and then their full last name. So for me (Peter Anthony O’Donnell), the output would be “Your name is PA O’Donnell”.

   To make this script work you will need to use the code block in the code block group.

5. **Boolean logic in Scribble: True or False?** As you have seen variables in Scribble can be used to store text. Another type of data that can me stored in a variable is Boolean data. Named after George Boole, Boolean variables hold only two possible values - true or false. Boolean expressions are expressions that evaluate to either true of false. Boolean variables and expressions are very important in computer software - in coming weeks we will use if-then code blocks to build scripts that make decisions. Boolean logic is the foundation of these programming structures. Boolean variables are simple to make in Scribble.

   Start a new project, and in the default sprite (Sprite1) make the following very simple script. Note you can make a Boolean variable by simply dragging and dropping either the **true** or **false** reporter from the Operators code block group into a set code block. You will need to create two variables, named p and q. (There is kind of a tradition when demonstrating logic truth tables to use p and q as variable names.)
When you run the script, each of the say block will display a result for 1 second. These demonstrate some basic logical operations. (The results should be true, false, false, true, false).

There are times when we will set Boolean variables to have simple values like true or false in a program (for example, as a status marker to show whether or not a processing stage has been completed), but more typically we will evaluate some other expression to a true or false value. Lets demonstrate that with the following script. Create this script (and the two variables it uses named number1 and number2). You will see that the result is false as number1 and number2 are not equal.

Most programming languages have a number of these logical comparison operators - you found the one you needed here in the Operators code block group. You'll notice there is also and operator for greater than > and for less than < comparisons.

Can you perform a greater than (or less than) and equals to comparison? Most programming languages have a built-in operator for this. Scribble doesn’t but you can still do the comparison. How?
6. **Numbers in Scribble: A pay calculating program.**

Consider the flowchart diagram on the left (you should have briefly met flowcharts in the lectures and tutorials, no matter if you haven’t seen one before, they are pretty easy to follow). Read it carefully.

The following Scribble code implements the algorithm represented by that flowchart (with the addition of a code block to make the script start when the green flag is clicked).

```
when green_flag clicked
   ask "Enter the number of hours worked."
   set hours to answer
   ask "Enter the employee's hourly pay rate."
   set payrate to answer
   set grosspay to hours * payrate
   say "The employee's gross pay is $", grosspay
```

Make that program, and check that it runs as you expected. Test it by calculating the pay of a worker who has worked 38 hours at a pay rate of $21.50 per hour. Your program should produce the correct answer of $817.
7. A calculating a discount sale price. Use the following flowchart as the specification of a program. Create the program in Scribble.

![Flowchart](image)

8. Yet more flowcharts and applications. You should be getting the hang of it now. Revisit the tutorial exercises from this week. Make Scribble programs to implement the problems you worked on in tasks 1.1 through to 1.4.

---

1. Developing flowcharts. Draw flowcharts for each of the following problems (adapted from Gaddis, 2013, p77). If you have access to a computer … try to draw some by hand, and some using a software tool.

1.1. A company has determined that its annual profit is typically 23 percent of total sales. Create a flowchart for a program that asks the user to enter the project amount of total sales, and then displays the profit that will be made from that amount.

1.2. One acre of land is equivalent to 43,560 square feet. A square foot is equivalent to 0.092903 square metres. Create a flowchart for a program that asks the user to enter the total number of square metres of land, and then calculates and displays the number of square feet and acres in that tract.

1.3. A customer in a store is purchasing five items. Create a flowchart of a program that asks for the price of each item, then displays the subtotal of the sale, the amount of sales tax (GST) and the total. Assume GST is 10%.

1.4. Assuming there are no accidents or delays, the distance a car travels down the freeway can be calculated using the following formula:

\[ \text{Distance} = \text{Speed} \times \text{Time} \]

A car is travelling at 60 kmps per hour. Create a flowchart for a program that displays the following:

- The distance the car will travel in 5 hours
- The distance the car will travel in 8 hours
- The distance the car will travel in 12 hours
9. Some color fun. Enough of the focus on flowcharts and variables. Let's do something much more fun. Look at the following Scribble code. It's very similar to the code you would have seen your lecturer demonstrate in the lecture. It uses a few variables to define the repeated drawing of a shape (a rectangle). Create your own version of this application and verify that it works correctly. It should draw a circle of squares of different colours.

![Scribble code example]

Once you have it working. Play with it. Introduce more variables to vary other parameters like location, shading, and ghosting. Some things to try:

- Make the variables simply on the stage as sliders so you can interact with the script as it runs.
- Duplicate the script that responds to the “draw” message, or,
- Duplicate the sprite - perhaps have one drawing a rectangle, another a circle, and another an oval.
- You’ll get interesting effects also if you move the goto random location from the script that is used to initialise variables to the script that repeats the drawing.
- Enjoy, show what you can make to your class mates. Wander around and see what they have been able to create too.

10. That’s it - with about 30 minutes to go in the class your tutor will distribute the in-class test for this laboratory. This is worth 1% of your mark for the unit. It should only take 15 to 20 minutes to complete. You must work on it on your own. Give your answer sheet back to your tutor at the end of the class. You will get the result next week (don’t worry, it’s not hard).

Tasks (off-campus students)

1. Making a variable and giving it a value. Variables are an important part of any software application. It’s pretty easy to create a variable in Scribble. Start a new Scribble application. For the moment we are just going to make some simple scripts, that we will run by double clicking, in the default sprite (Sprite1). To create a variable just go to the code block group in the scripting toolbox. The first item there is a grey button labelled “Make a variable”. Click that button, a dialog appears asking you to specify a couple of options for the variable that is being created.
You should choose meaningful names for your variables. That will make your code easier to understand - esp. by others, and easier to debug when something is wrong. With that advice in mind, create a variable - called x! (Despite what I just wrote, it is common for variables that are used for simple counting or housekeeping purposes to be given simple names like x, y and z. It’s not great practice, but it’s common.)

Notice the other option available at the time you create a variable in Scribble allows for you to specify the scope of the variable. More on this in this week’s lecture and also later in the unit; variable scope is an important concept for you to understand. It’s a powerful tool to use to make sure the code you write in maintainable. In general, making variables “global” makes your work as a programmer easier in the short term and harder in the long term. It’s considered good practice to restrict the scope of variables as much as possible. Ignoring that, simply make the scope of this variable x global - that’s is available for all sprites. Click the OK button and the variable is created. You should immediately notice a couple of changes to the screen. The variable x should be added as a drag-able item in the code block group. You should also notice that the variable appears in the stage area. See the little check box next to the variable where it appears in the code block group. Click that check box to turn it off, and you’ll see the display of the variable in the stage area vanishes. Click it again and it reappears. That’s a neat device for showing the values of variables while your code is running, and then once you have made sure your code is working correctly you can remove them from the main display making it look cleaner.

1. When a variable is created it appears here in the Variables code block group.
2. The variable name and its current value appears here in the stage.
3. Toggle the check box next to the variable here (see 4).
4. With the check-box unselected the variable and its value are no longer shown.
2. A look back at the lilac chaser: generalising code with variables. Do you have a copy of the lilac chaser application you created in laboratory 1? If you do open that now. If not, download a copy from the unit web site. Add a variable to that application, and name it `waittime`. Make it's scope global, so that it is available in all sprites. In the script that belongs to the canvas - the one that reacts to the click of a green flag and starts the process of the dots flashing by broadcasting a message named “show dot 1” add a set code block that sets the value of the variable `waittime` to 0.1.

Now, in each of the dots, find the `wait` command in the each of the scripts. Drag the variable `waittime` (from the Variables code block group) into the body of the `wait` command - replacing the value 0.1 that you typed there with the variable `waittime`.

Once you have done that, do it for all the dots (sprites) in your application. As `waittime` has a value of 0.1, that value will be the time (in seconds) that is used in executing the `wait` command so the application should run just the same as before after you make all the changes. Verify that this is the case.

Use the on-line discussion forum to discuss why you were asked to do that. It was more work. Are there any benefits? This section was headed ... generalising your code ... what did that mean?

2.1. It's now possible by changing the value of `waittime` in the script that belongs to the canvas (where it's value is initialised) to give it a value other than .1 and for all the flashing of the dots in the circle to speed up or slow down. try that with a few different values, try 0.01 and 1.0. After you have done that, set the value back to .1. Now use the click box next to the variable name (on display in the Variables group of code blocks) to turn on the display of the variable in the stage area. Click on the variable there. You'll notice that is cycles through thee different forms of display. The default which shows the variable name and it's value, a slider that has an interactive slider bar under the number and a final
display that just shows the value and not the name of the variable. Click to make the slider bar form appear. Then right click on the slider bar.

1. Double click on the variable display on the stage and the “type” of display changes.

2. First time is shows a slider. This is interactive, manipulating the slider changes the variable’s value.

3. Second time it removes the variable’s name and just shops the value. Double click again and it goes back to 1.

4. When the slider view is selected, right clicking on the item displays a pop-up menu. From this menu you can set the min and max value of the slider.

3. Basic input and output with Scribble. The say and ask commands. We are about to make one of the dumbest programs you could imagine - it’s standard fare in introductory courses on programming - so please forgive me. It’s a program that asks an end user for their name and then just prints (using the say command) it back to them. We are going to create a version of it in Scribble to illustrate the use of the ask and say commands, the answer variable and the join operator. All important and simple command blocks that in a shirt time you will be able to use without thinking. Start a new project, and in the default sprite (Sprite1) create the following script.

Begin by making a variable called name. Then start a script using the ask command block from the Sensing command block group. Connect a set command to that, make the focus of the set command the variable name. Go back to the Sensing command block group, you’ll notice there is a built-in variable there named answer. This built-in variable stores the value of whatever input the user has provided when an ask code block runs. Drag that answer variable into the editable part of the set command (the result when it runs will put the user input into the name variable). Now, snap a say command to the bottom of the script (from the . The say command in Scribble is used for simple display of results. The “thing” that is output can be text or the result of the evaluation of some expression. The output we want for this script is going to combine two pieces of text - the text “Your name is ” and the value stored in the variable name. To do that we will use a join operator. Look for that in the code block group. Drag the join operator into the say command. Then type the text “Your name is ” into the first part of is, then drag the variable name into the second part. After that test your script by double clicking on it. When prompted by the sprite, type in your name and the script should then display the result.
The script uses all the commands mentioned earlier: ask, say, set, the join operator and the built-in variable answer. None of that was hard, but it is really important that you understand how those commands work as you will use them a lot. This is the result of running the script when I typed in my name (Peter). Your's should work in a similar manner.

Even though that's a simple script, let's explore it a bit more. Let's consider for the moment the join operator. It joins two items of text together. The text items might be something typed in to the operator directly (like the text “Your name is”), or a value in a variable, or the result of the evaluation of an expression. One of the things that can be joined - is the result of another join. This is a technique (often used in programming) called nesting. By nesting join statements you can make text output that is made up of any number of items. Let's change our script to include nested join operators in the say command.
Nesting commands like that is very useful. It allows simple program elements to create much provide complex functionality. It can, however, make program code hard to follow. One way of helping keep the scripts you write readable (and as a result easier to debug and later to maintain) is to break up expressions (not just ones that are nested) and store intermediate results in variables. Here is an example of a script - not that two joins nested is hard to read - that does the same work as the previous script but its separates the two joins. The say command simply outputs the value stored in the variable *greeting*. 

1. This say command has been completed by using two join statements. One dropped inside the other. This is called *nesting*.

2. This is the output of that script if the end-user has typed “Peter” in response to the prompt “What’s your name?”

Nesting commands like that is very useful. It allows simple program elements to create much provide complex functionality. It can, however, make program code hard to follow. One way of helping keep the scripts you write readable (and as a result easier to debug and later to maintain) is to break up expressions (not just ones that are nested) and store intermediate results in variables. Here is an example of a script - not that two joins nested is hard to read - that does the same work as the previous script but its separates the two joins. The say command simply outputs the value stored in the variable *greeting*.

make your script like that. You'll need to create two variables: *salutation* and *greeting*, and add some more set code blocks to your script. Verify that your script works as before.

Another point not to be lost is the flexibility of the say code block. It can output text that has been typed in, the value of a variable, or it can evaluate an expression and output the result. In a moment we will create some boolean expressions and use the say command to show the results.

4. **String manipulation.** Build on the script you have just made. Make the script ask the user for their firstname, their middle name and their last name. Then output the sentence “Your name is Name”. Make Name be the first initial, their second initial a space and then their full last name. So for me (Peter Anthony O’Donnell), the output would be “Your name is PA O’Donnell”.

To make this script work you will need to use the code block *letter 1 of word* in the Operators code block group.

5. **Boolean logic in Scribble: True or False?** As you have seen variables in Scribble can be used to store text. Another type of data that can me stored in a variable is Boolean data. Named after
George Boole, Boolean variables hold only two possible values - true or false. Boolean expressions are expressions that evaluate to either true or false. Boolean variables and expressions are very important in computer software - in coming weeks we will use if-then code blocks to build scripts that make decisions. Boolean logic is the foundation of these programming structures. Boolean variables are simple to make in Scribble.

Start a new project, and in the default sprite (Sprite1) make the following very simple script. Note you can make a Boolean variable by simply dragging and dropping either the true or false reporter from the Operators code block group into a set code block. You will need to create two variables, named p and q. (There is kind of a tradition when demonstrating logic truth tables to use p and q as variable names.)

When you run the script, each of the say block will display a result for 1 second. These demonstrate some basic logical operations. (The results should be true, false, false, true, false).

There are times when we will set Boolean variables to have simple values like true or false in a program (for example, as a status marker to show whether or not a processing stage has been completed), but more typically we will evaluate some other expression to a true or false value. Let's demonstrate that with the following script. Create this script (and the two variables it uses named number1 and number2). You will see that the result is false as number1 and number2 are not equal.

Most programming languages have a number of these logical comparison operators - you found the one you needed here in the Operators code block group. You'll notice there is also an operator for greater than > and for less than < comparisons.

? Can you perform a greater than (or less than) and equal to comparison? Most programming languages have a built-in operator for this. Scribble doesn't but you can still do the comparison. How? Let's see who is the first to work it out and post their answer on the discussion forum.

Consider the flowchart diagram on the left (you should have briefly met flowcharts in the lectures and tutorials, no matter if you haven’t seen one before, they are pretty easy to follow). Read it carefully.

The following Scribble code implements the algorithm represented by that flowchart (with the addition of a code block to make the script start when the green flag is clicked).

```
when [clicked]
  ask Enter the number of hours worked and wait
  set hours to answer
  ask Enter the employee’s hourly pay rate and wait
  set payrate to answer
  set grosspay to hours * payrate
  say join The employee’s gross pay is $, grosspay
```

Make that program, and check that it runs as you expected. Test it by calculating the pay of a worker who has worked 38 hours at a pay rate of $21.50 per hour. Your program should produce the correct answer of $817.
7. A calculating a discount sale price. Use the following flowchart as the specification of a program. Create the program in Scribble.

8. Yet more flowcharts and applications. You should be getting the hang of it now. Revisit the tutorial exercises from this week. Make Scribble programs to implement the problems you worked on in tasks 1.1 through to 1.4.

1. Developing flowcharts. Draw flowcharts for each of the following problems (adapted from Gaddis, 2013, p77). If you have access to a computer ... try to draw some by hand, and some using a software tool.

1.1. A company has determined that its annual profit is typically 23 percent of total sales. Create a flowchart for a program that asks the user to enter the project amount of total sales, and then displays the profit that will be made from that amount.

1.2. One acre of land is equivalent to 43,560 square feet. A square foot is equivalent to 0.092903 square metres. Create a flowchart for a program that asks the user to enter the total number of square metres of land, and then calculates and displays the number of square feet and acres in that tract.

1.3. A customer in a store is purchasing five items. Create a flowchart of a program that asks for the price of each item, then displays the subtotal of the sale, the amount of sales tax (GST) and the total. Assume GST is 10%.

1.4. Assuming there are no accidents or delays, the distance a car travels down the freeway can be calculated using the following formula:

\[ \text{Distance} = \text{Speed} \times \text{Time} \]

A car is travelling at 60 km\text{hr}\text{.} per hour. Create a flowchart for a program that displays the following:

- The distance the car will travel in 5 hours
- The distance the car will travel in 8 hours
- The distance the car will travel in 12 hours
9. **Some color fun.** Enough of the focus on flowcharts and variables. Let's do something much more fun. Look at the following Scribble code. It's very similar to the code you would have seen your lecturer demonstrate in the lecture. It uses a few variables to define the repeated drawing of a shape (a rectangle). Create your own version of this application and verify that it works correctly. It should draw a circle of squares of different colours.

```scribble
when I receive draw
  turn 90 degrees
  move 100 steps
  change fill color by 40
  rectangle 50 by 50
  broadcast draw
```

Once you have it working, play with it. Introduce more variables to vary other parameters like location, shading, and ghosting. Some things to try:

- Make the variables simply on the stage as sliders so you can interact with the script as it runs.
- Duplicate the script that responds to the “draw” message, or,
- Duplicate the sprite - perhaps have one drawing a rectangle, another a circle, and another an oval.
- You’ll get interesting effects also if you move the goto random location from the script that is used to initialise variables to the script that repeats the drawing.
- Enjoy, show what you can make to your class mates. Wander around and see what they have been able to create too.

10. On the “study guide” tab of the unit-web site for this week you will find a link to this week’s on-line test for the semester. This is worth 1% of your mark for the unit. It should only take 15 to 20 minutes to complete. You must work on it on your own. The test will be available for you to do for 1 week (you won’t be able to access it after midnight on Sunday). If you have any problem accessing the test please send Peter O’Donnell (peter.odonnell@monash.edu) an email.

---

**Further reading and information**

Wikipedia entries on: