IMAT1604 Visual Web Development

Contents

Using the Graphical Programming Environment

What is GPE?

GPE is a simple programming language which allows you to enter a set of commands to draw pictures and patterns. It introduces some key concepts in programming for example variables, objects, methods, properties, sequence, assignment and functions.

Preparing for the Assessment

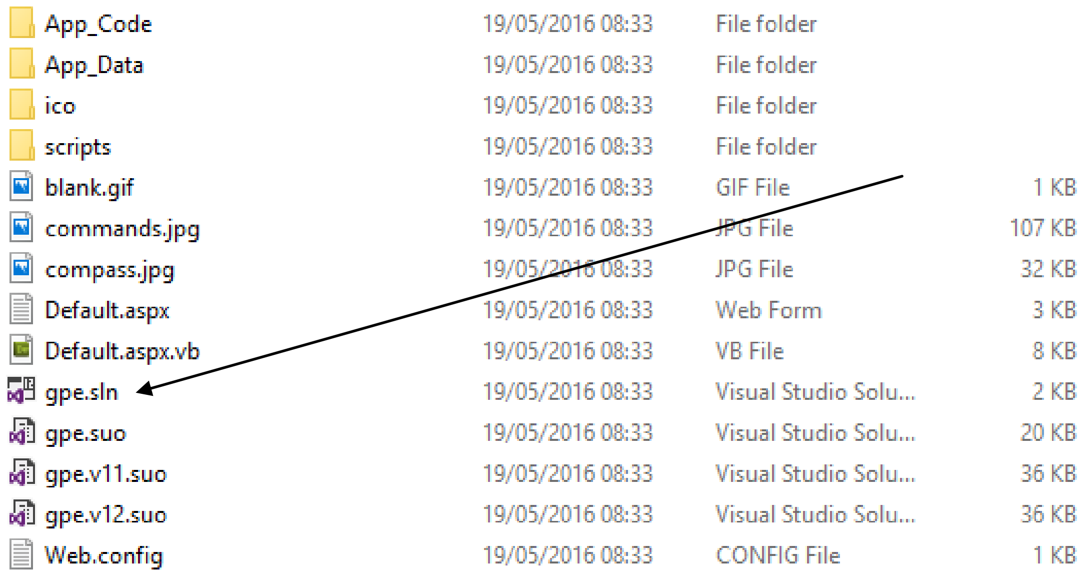
As part of the assessment for this module you will be required to complete a few things. It is worth looking at the assessment requirements now so that you may start thinking about them.

For this session you are required to complete individually a mock sprint log. The log needs to include a few words about the work you have completed in the lab and handed in to your tutor. They will feed back to you next week to make sure that you have got the hang of completing the paperwork for the module.

Starting GPE - Opening your First Visual Studio Project

The first step is to download and extract the files for the GPE program. The file “GPE Web.zip” is available from the module web site.

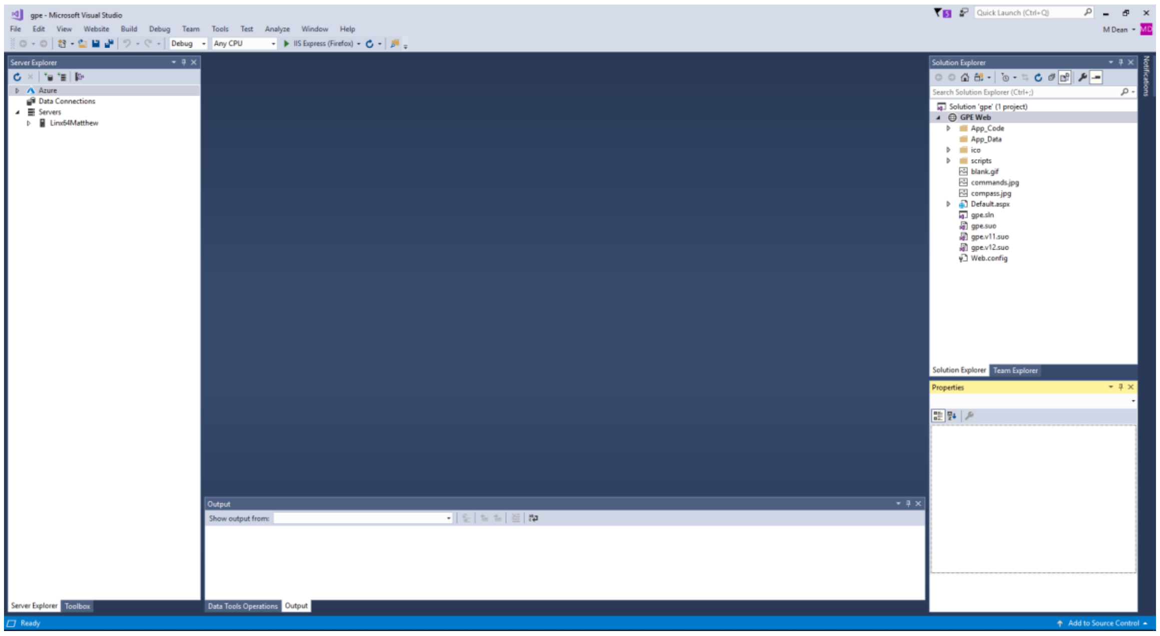
Once you have extracted the files you should see a file called gpe.sln



App_Code	19/05/2016 08:33	File folder	
App_Data	19/05/2016 08:33	File folder	
ico	19/05/2016 08:33	File folder	
scripts	19/05/2016 08:33	File folder	
blank.gif	19/05/2016 08:33	GIF File	1 KB
commands.jpg	19/05/2016 08:33	JPG File	107 KB
compass.jpg	19/05/2016 08:33	JPG File	32 KB
Default.aspx	19/05/2016 08:33	Web Form	3 KB
Default.aspx.vb	19/05/2016 08:33	VB File	8 KB
gpe.sln	19/05/2016 08:33	Visual Studio Solu...	2 KB
gpe.suo	19/05/2016 08:33	Visual Studio Solu...	20 KB
gpe.v11.suo	19/05/2016 08:33	Visual Studio Solu...	36 KB
gpe.v12.suo	19/05/2016 08:33	Visual Studio Solu...	36 KB
Web.config	19/05/2016 08:33	CONFIG File	1 KB

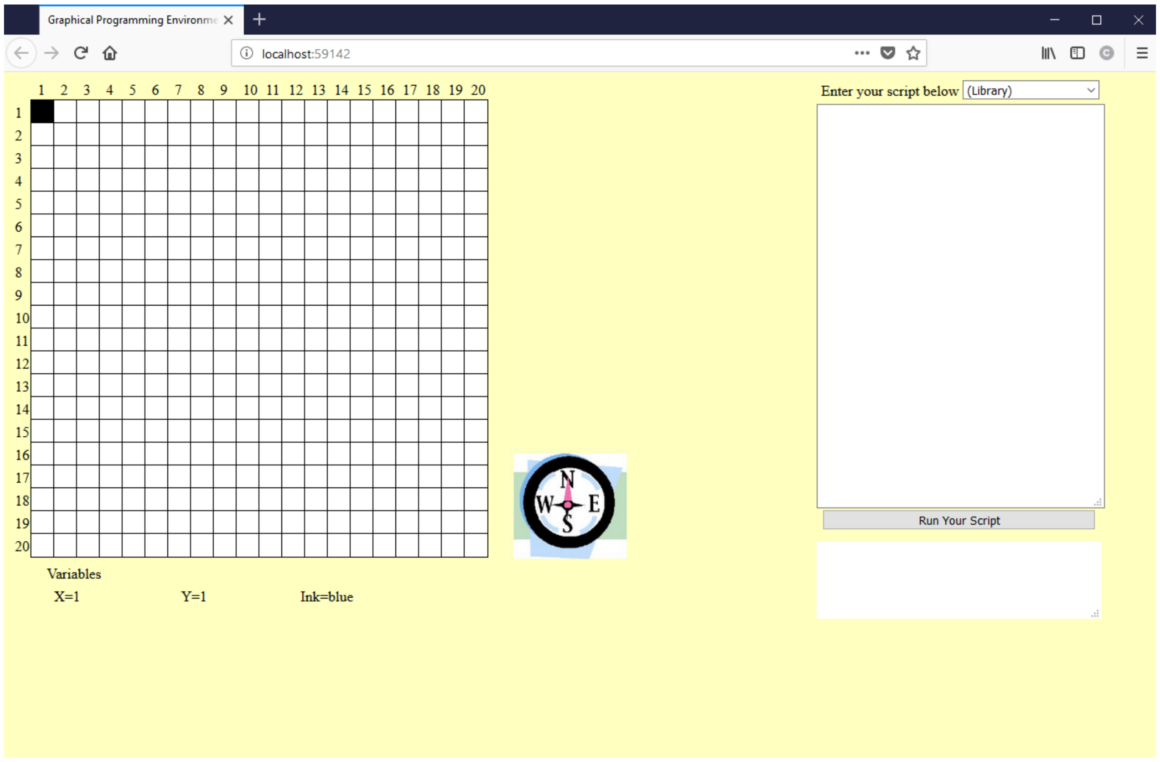
To open the web site, double click this file to open the site in Visual Studio.

If all goes to plan you should see something like this...



To run the program and view it in the browser press F5 on the keyboard.

You should see something like this …



Variables

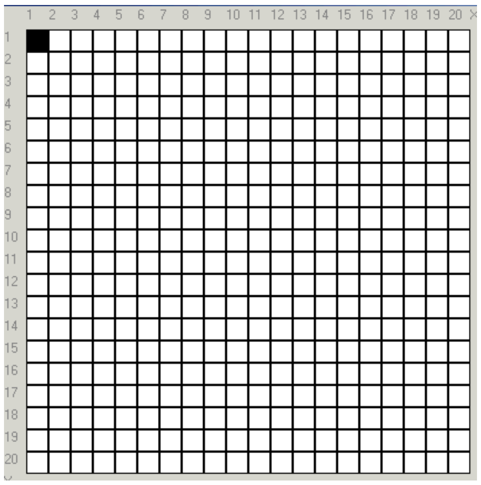
X=1 Y=1 Ink=blue

Overview of the GPE Interface

There are four main sections to GPE

- the Screen
- the Script Box
- the Variables Display

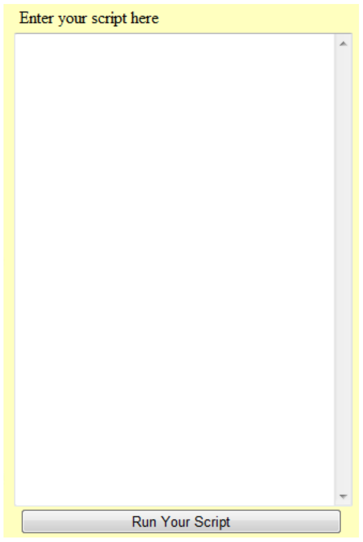
The Screen



This screen is where your picture will be drawn. It consists of a 20 X 20 grid.

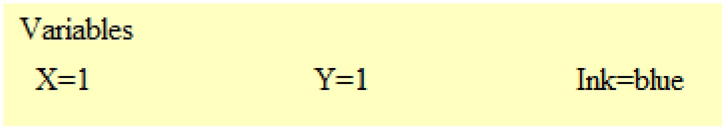
Note the black square in the top left. This tells you where the pen is and therefore where the drawing will start from.

The Script Box



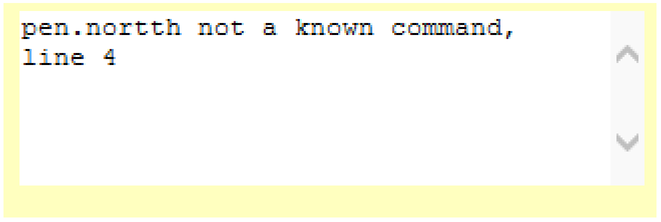
The script box to the right of the screen allows you to enter your commands to display a picture.

The Variables Display



This tells you the setting of different variables, namely the X and Y coordinates and current ink colour of the pen.

The Error List



```
pen.nortth not a known command,
line 4
```

This will provide you with error messages should something go wrong. People new to programming tend to shy away from error messages (perhaps it is easy to feel like you have failed when you get an error message?)

Error messages are however an essential part of the programming process. Experienced programmers generate errors. Without suitable error messages the programmer is in the dark as to what is going wrong with their code.

Never underestimate the value of error messages whilst programming!

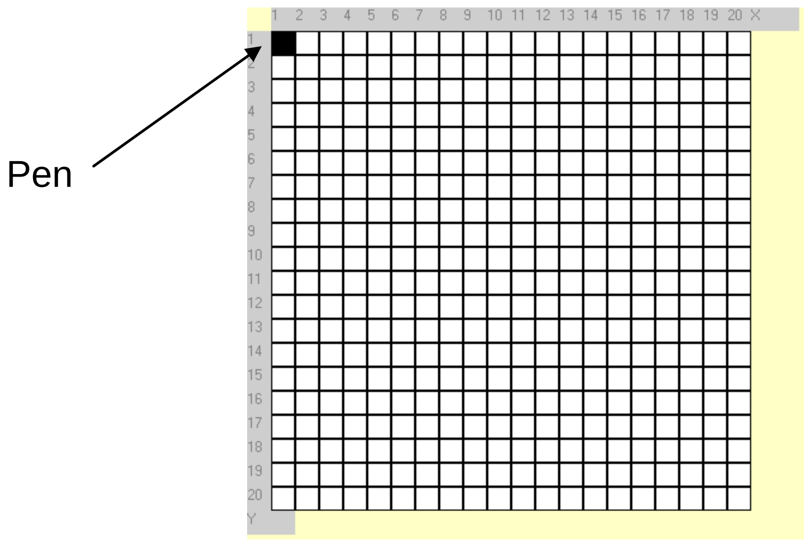
Objects in GPE

To control a computer system, we break the different components of the computer into objects.

Objects are simply words that we may use to describe and send commands to parts of the computer system.

Using our code, we interact with these objects be means of the object's methods and properties.

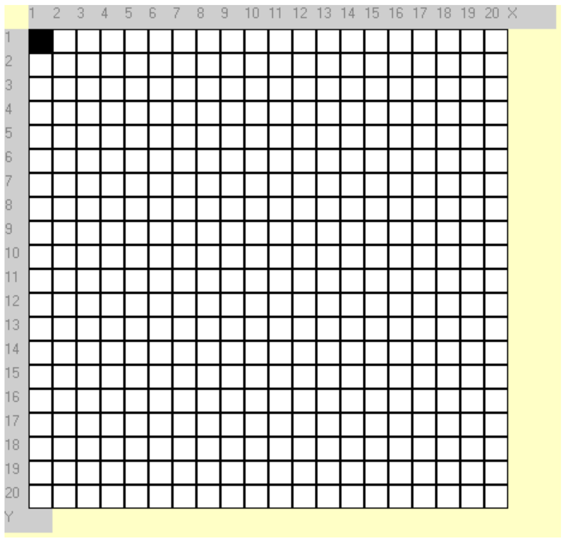
The main object within GPE is the Pen indicated by a black square…


Pen
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 X
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 Y

By setting this object’s properties and using its methods we may make the program move the pen and draw pictures.

- Methods tell an object to do something.
- Properties are used to change or read the settings of an object.

Another object in GPE is the screen.



This grid may also be controlled by using its methods.

GPE Methods and Properties

Below is a reference list of the objects methods and properties within GPE.

<u>Object Name : pen</u>
Methods

- .north (*{Distance})*
- .south (*{Distance})*
- .east (*{Distance})*
- .west (*{Distance})*
- .northeast (*{Distance})*
- .northwest (*{Distance})*
- .southeast (*{Distance})*
- .southwest (*{Distance})*
- .jumpto (*{X}*,*{Y}*)

Properties

- .ink = *{Colour}*

<u>Object Name : screen</u>
Methods
    .clear
Properties
    None

Syntax

To start writing a script we need to be aware of the syntax for the code.

Syntax is about arranging the words and symbols in a language so that they make sense. With a computer, this not only applies to the words in our instructions but the punctuation as well. Miss out even a comma or a space and the program may not work.

This is important when "talking" to a computer as a computer is too stupid to work out what you mean.

When writing a command in GPE we first need to identify which object we are sending the command to and then state the command to apply. These two items of text must be separated by a full stop.

For example, if we want to send the pen south we would do the following…

```
pen.south
```

Unfortunately, with most languages it is quite so simple!

The Semi Colon

In the case of GPE it is based on the C#. The letter C means that it is based on a language called C. The semi colon in C type languages is used as important punctuation.

So to enter the command pen.south it must have a semi colon like so…

```
pen.south;
```

If you miss the semi colon off (and you will) the program won't work.

Case Sensitivity

The other problem with many computer languages is that many are case sensitive.

```
Pen.south;
```

This won't work due to the capitalised letter P.

Entering your first Script

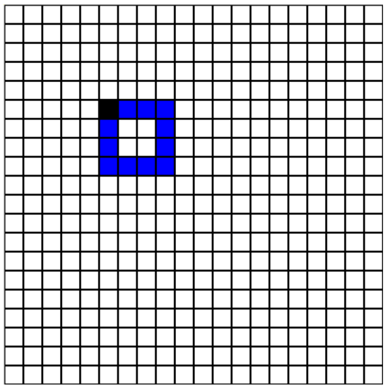
In the script box enter the following commands (jumpto is all one word with no space, welcome to the world of fussy syntax!)

```
pen.jumpto(6,6);
screen.clear;
pen.east;
pen.east;
pen.east;
pen.south;
pen.south;
pen.south;
pen.west;
pen.west;
pen.west;
pen.north;
pen.north;
pen.north;
```

Once you have typed in all the commands, press the Run button to run the script.

You have just written your first computer program. Although these are just words on the page they are words that when presented to it means something to the computer.

You should see a blue square in the output grid.



Sequence

Each command is run one after the other.

This is called <u>sequence</u>. The sequence runs from line 1 to line n. It doesn't go backwards, it doesn't skip over lines unless we deliberately break the rules of sequence.

The first method *jumpto(6,6)* forces the pen (the black square) to jump to the grid reference 6 , 6. (X , Y)

Notice that the x and y coordinates are placed in brackets. This indicates that these are "parameters" which tell the jumpto command what it needs to know.

screen.clear clears the screen so that we are starting with a blank grid.

The next commands move the pen in the specified direction, so…

```
pen.east;
pen.east;
pen.east;
```

Moves the square east by three squares.

Moving south then west then north, each by three squares takes the pen back to the start and completes the square.

The Order of Commands is Important!

When writing code in any language, it is important to understand that the order in which you enter your commands is important.

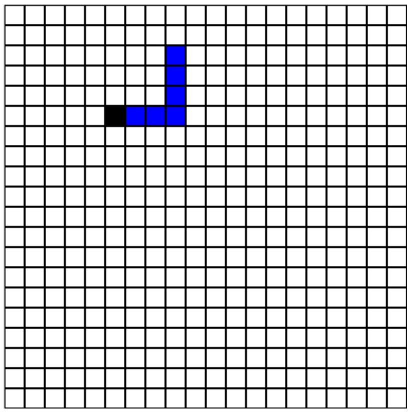
If we use the same commands but change the order, we don’t get a blue square!

Try the following…

(Delete the text of the previous script before you enter the one below!)

```
pen.jumpto(6,6);
screen.clear;
pen.east;
pen.east;
pen.east;
pen.north;
pen.north;
pen.north;
pen.south;
pen.south;
pen.south;
pen.west;
pen.west;
pen.west;
```

Rather than a blue square, you will get a blue reverse L…



The same commands were present, but by changing the order, the result changed.

Learn to be Concise

Programming languages provide a large range of features, many of which should help to make your life simpler.

We could simplify the blue square script like so…

```
pen.jumpto(6,6);
screen.clear;
pen.east(3);
pen.south(3);
pen.west(3);
pen.north(3);
```

Rather than writing east three times, adding the parameter onto the command makes the command repeat three times. The use of parameters in the method allows us to specify how many times the method should repeat.

It is now a much neater script, reduced from fourteen lines down to six.

Learning to be concise with code means that your programs run faster as there are fewer lines to run, also the programs will be smaller as each line of code takes up memory and storage space. It also makes it a lot easier to find the inevitable problems with our code.

Overview of the Pen’s Methods and Properties

North, South, East, West, NorthEast, NorthWest, SouthEast and SouthWest

Each of these methods moves the pen in the specified direction. By adding a number after the method, it will repeat the command that number of times, so…

pen.south(10) will send the pen down the grid 10 squares.

pen.southeast(5) will send the pen diagonally for five squares.

.ink=

The ink property allows you to change the colour of the ink, i.e. the colour left behind by the pen.

Acceptable values are black, red, blue, green and white.

So, the command…

```
pen.ink = green;
```

…will result in the next line drawn by the pen to be drawn in green.

The = symbol (assignment operator) is used to set the value of the ink.

jumpto(X,Y)

This method forces a jump to the specified coordinates on the screen.

pen.jumpto (10,15) for example will move the pen directly to the X coordinate 10 and the Y coordinate 15

Double Slash //

The double slash may be used to insert comments for example…

```
//this is a square
screen.clear;
pen.jumpto(6,6);
pen.east(3);
pen.south(3);
pen.west(3);
pen.north(3);
```

The first line //this is a square – is ignored by the program but tells you the user what the following lines do.

Exercises

Exercise 1

To see that you have the idea of how the program works the first exercise is simple.

To start with, draw a square anywhere on the grid 6 cells by 6.

Once you have drawn the square modify your script so that the colour of the square is red.

Exercise 2a

Create a script that draws two equal size rectangles. Each rectangle must be as large as possible but must not touch the other or the sides of the output grid. The rectangles must be wider than they are tall. The top rectangle should be blue, the bottom one red. Neither rectangle should be filled.

Points for Exercise 2a

Understand the Problem Clearly

Compare your results for Exercise 2a with another student in your lab group. Do you have the same solution? If not, why not?

How important is it that you clearly understand what is required of you?

With any programming that you are going to perform there will be some sort of requirements that need to be met.

The requirements may be…

“Write a web site allowing the sale of books over the Internet.”

“Create a database that stores student results.”

The starting point of all programs is the idea of some task or problem to be addressed.

If you are clear on the initial requirements of your program then your system will be off to a good start.

Get it wrong at this early stage and you will not meet the requirements. This may mean you fail your project or don’t get paid!

Exercise 2b

Create a script to draw two squares as large as possible, one inside the other. Neither square may touch the other nor may they touch the sides of the output grid. The inside square should be green, the outside square black. Neither square should be filled. Before entering the script into the program, plan the squares in pencil using the grid below.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1																				
2																				
3																				
4																				
5																				
6																				
7																				
8																				
9																				
10																				
11																				
12																				
13																				
14																				
15																				
16																				
17																				
18																				
19																				
20																				

Was exercise 2b any easier than exercise 2a? If so why?

Plan Your Design

Even with a simple language and simple tasks like the ones above, you will probably appreciate that a little planning made the task of writing the script a lot easier.

Take time when programming to plan what you want to do and how your design will work.

There are many tools available to help programmers design their system, but even without such tools there is always pen(cil) and paper.

Planning and documentation become even more important especially when the program is large and complex, also when there are large numbers of people working on the system.

Exercise 3

Open the script …

h.txt

The script should draw a large H on the screen.

What happens when you run it?

Programming Languages are Precise with Precise Syntax

In the above script, the line…

```
pen.North (6)
```

…was mistyped resulting in an error.

If you haven’t done so already, correct the error and see the program running.

The sad fact is that programming languages are very fussy about the syntax of the commands.

Consider the jumpto command.

```
pen.jumpto(6,6);
```

This is a valid command.

```
pen.jumptoo (6,6);
```

This is not valid because JumpTo is spelt wrong.

```
pen.jumpto (6 6);
```

This is not valid because the comma is missing.

```
pen.jumpto (200,200);
```

Is not valid because the grid is only 20 by 20!

The commands we type into a program must make logical sense, in that we can’t jump to a cell that doesn’t exist.

```
Pen.jumpto(6,6);
```

Is not a valid command P is capitalised.

Commands must also be spelt and set out correctly and computers are really, fussy!

Each command in any programming language will have a set syntax or layout.

Get this wrong and the command won't work.

Programming Languages are Stupid

Also note that in the above example, even though the script contained an error, the program still did exactly as it was told.

If the code you create doesn't make sense, don't expect the computer to "work out" what you are getting at!

For the next two exercises, you will need to time how long it takes you to complete each task.

Exercise 4a

Open the file

house1.txt

Notice that the windows are blue, locate the command that does this and change them to red. Use a watch to time yourselves while you do this.

Exercise 4b

Open the file

house2.txt

Notice that the grass is blue, locate the command that does this and change it to green. Use a watch to time yourselves while you do this.

Questions for Exercise 4

How did the two tasks compare in complexity?

What made the difference between exercises A and B?

All programming languages have some mechanism to type comments into the code.

Comments don’t serve any function for the computer, but for the human beings reading the code they are very useful if not essential.

Compare the time it took to problem solve the commented script compared with the uncommented script.

If you write code that other people are going to amend always add comments so that they understand what you were doing.

If you write code that others will never see, still add comments! It is amazing how you forget how your own code works when you come back to it after a few months.

Functions in GPE

One thing you will eventually find with any programming language is that at some point you will find that it doesn't perform certain tasks automatically.

Programming languages have the facility to add extra functionality to your code by allowing you to write your own functions.

Functions allow you to create a small "program" which draws a frequently used shape, then when you need that shape you call the function to invoke it.

For example, enter and run the following script…

```
pen.jumpto(2,2);
screen.clear;
pen.east(5);
pen.south(5);
pen.west(5);
pen.north(5);
```

…it will draw a square on the grid.

Now imagine that you wanted four more squares all of the same size, you could enter (try the following script yourself.)

```
//square 1
pen.jumpto(2,2);
Screen.clear;
pen.east(5);
pen.south(5);
pen.west(5);
pen.north(5);

//square 2
pen.jumpto(10,2);
pen.east(5);
pen.south(5);
pen.west(5);
pen.north(5);

//square 3
pen.jumpto(2,9);
pen.east(5);
pen.south(5);
pen.west(5);
pen.north(5);

//square 4
pen.jumpto(10,9);
pen.east(5);
pen.south(5);
pen.west(5);
pen.north(5);
```

Ok, that's one possible solution, but what if we wanted ten squares?

An even more important question, what if we wanted to change the size of all the squares by one cell?

Try making all the squares in the above script 6 X 6 rather than 5 X 5.

Using functions, we could create a script for our square and then reuse it as and when required.

For example,…

```
function square
{
pen.east(5);
pen.south(5);
pen.west(5);
pen.north(5);
}
```

…defines the function.

Notice that the function is started by the word *function* and then the name of the script, in this case the name *square*. The function must also be enclosed with braces {}.

To call the function procedure the *call* key word is used.

So…

```
call square;
```

…will draw the square.

but before we do that it is a good idea to specify where the square should be placed with a jumpto command.

So…

```
pen.jumpto(2,2);
call square;
```

will draw a square at cell 2 , 2.

The full script for you to try is…

```
pen.jumpto(2,2);
screen.clear;
call square;
end script;

function square
{
pen.east(5);
pen.south(5);
pen.west(5);
pen.north(5);
}
```

The full script to draw the four squares would read…

```
//square 1
pen.jumpto(2,2);
screen.clear;
call square;

//square 2
pen.jumpto(10,2);
call square;

//square 3
pen.jumpto(2,9);
call square;

//square 4
pen.jumpto(10,9);
call square;

end script;

function square
{
pen.east(5);
pen.south(5);
pen.west(5);
pen.north(5);
}
```

To modify the size of all four squares you would change the square function.

Using the above script, modify it so that the square is 6 X 6. How did this task compare with the previous script?

Exercise 5

To illustrate the above, run the script ...

noughts and crosses1.txt

Modify the script to complete the game.

Now try the same task using the script

noughts and crosses2.txt

Avoid Repeating Code

As you can see in the above example, creating duplicated code makes maintenance and modification much more difficult. With any programming language think about how to arrange your code so that code blocks may be re-used. This makes your programs smaller and makes it easier to make changes.

Exercise 6

Run the script

house3.txt

Compare it with

house4.txt

Try modifying the walls to blue on each script.