COMP5110
Multimedia Development

Assignment 2

Gibson Lam

Graphic Programming

- This assignment covers two different parts of graphic programming:
  - Part 1:
    Making textures using Perlin noise data
  - Part 2:
    Growing a tree using an extended L-system
- Different techniques have been used in these two parts so you can work on each of them separately

The Assignment System

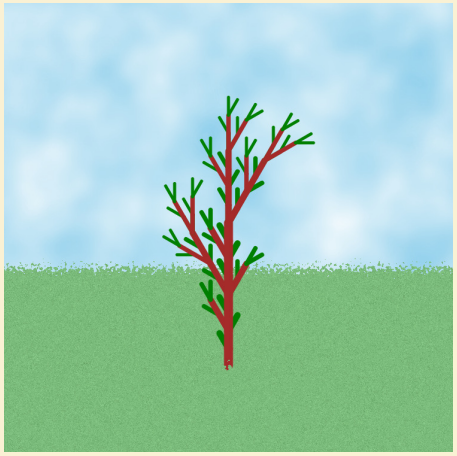
- Here is an example display of the assignment:

The Starting System

- The starting system is available in course canvas website
- The system has a similar structure as the one you have done for assignment 1
- However, this time you will need to use more JavaScript specific techniques when you work on the assignment

The SVG Area

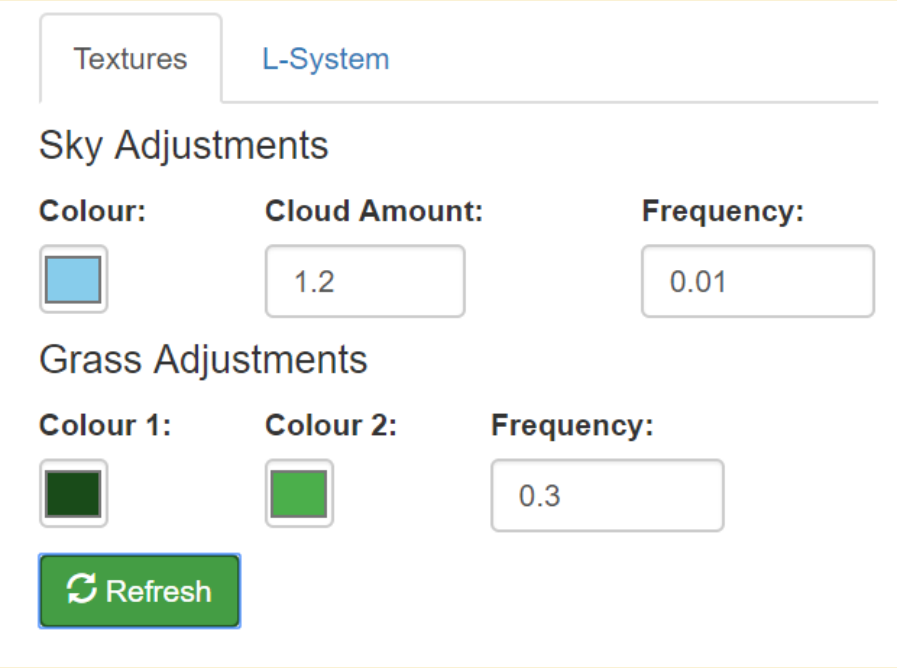
- The SVG area is an embedded SVG inside an HTML file
- The size of the SVG is 500 by 500 pixels



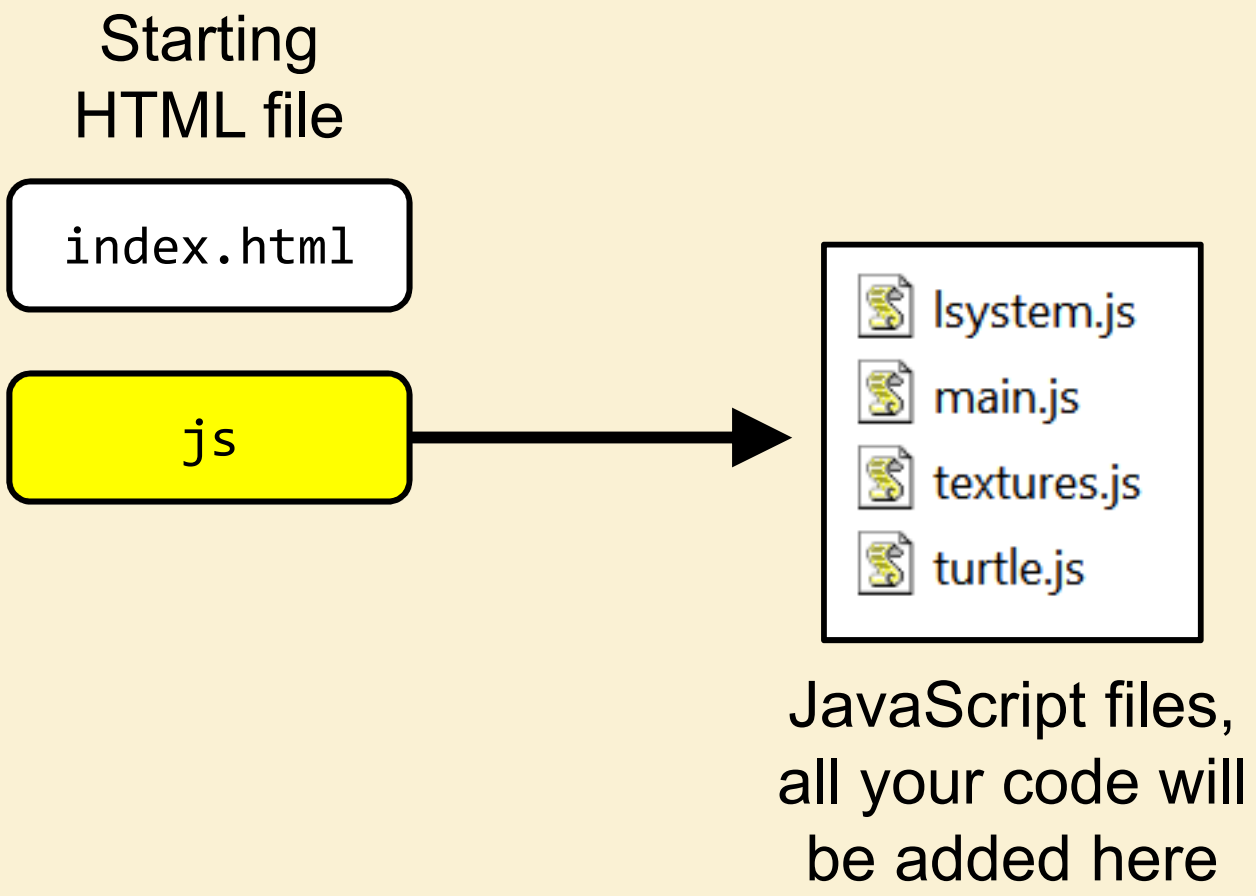
```
<svg width="500" height="500"
      xmlns="http://www.w3.org/2000/svg">
    <defs>
        ...define your filters here...
    </defs>
    ...SVG content here...
</svg>
```

The Control Area

- The control area is the place where you can change the settings of the textures or L-system parameters

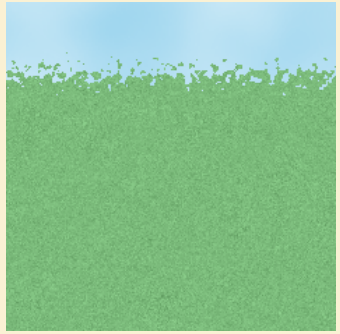


File Structure



Assignment

- For the first part of the assignment, you need to build a couple of Perlin noise textures that are used in a few SVG elements
- Here are the two textures that you need to make:
  - The sky texture
  - The grass texture



The Rectangle SVG Elements

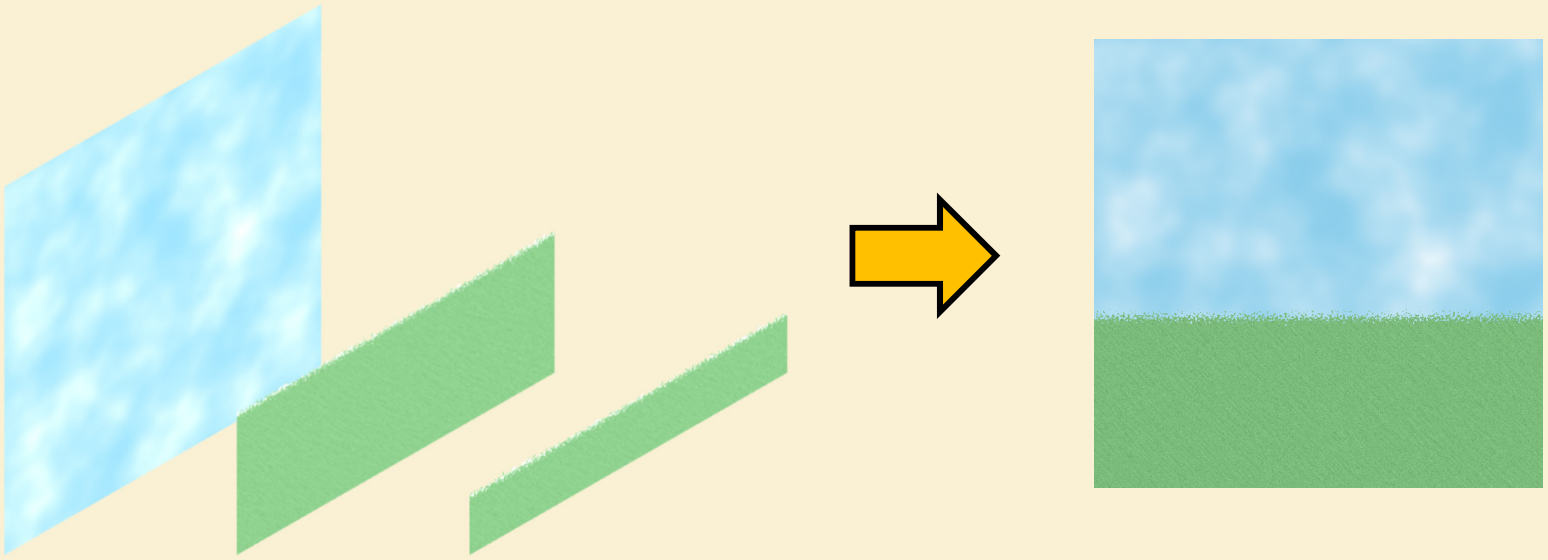
- There are three rectangle SVG elements given in the starting system:

```
<rect x="0" y="0" width="500" height="500"
      filter="url(#sky)"/>
<rect x="0" y="300" width="500" height="200"
      filter="url(#grass)"/>

<g id="tree"/>

<rect x="0" y="400" width="500" height="100"
      filter="url(#grass)"/>
```

The group is not relevant for this part of the assignment; it is for part 2: L-system

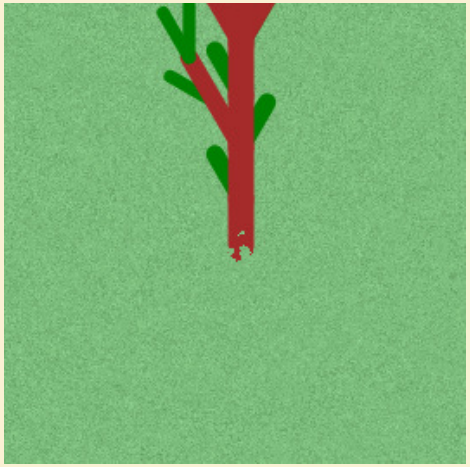
The Rectangle Arrangement

- By looking at the SVG code, you can see that the rectangles are arranged in three layers like this:



The Grass Layers

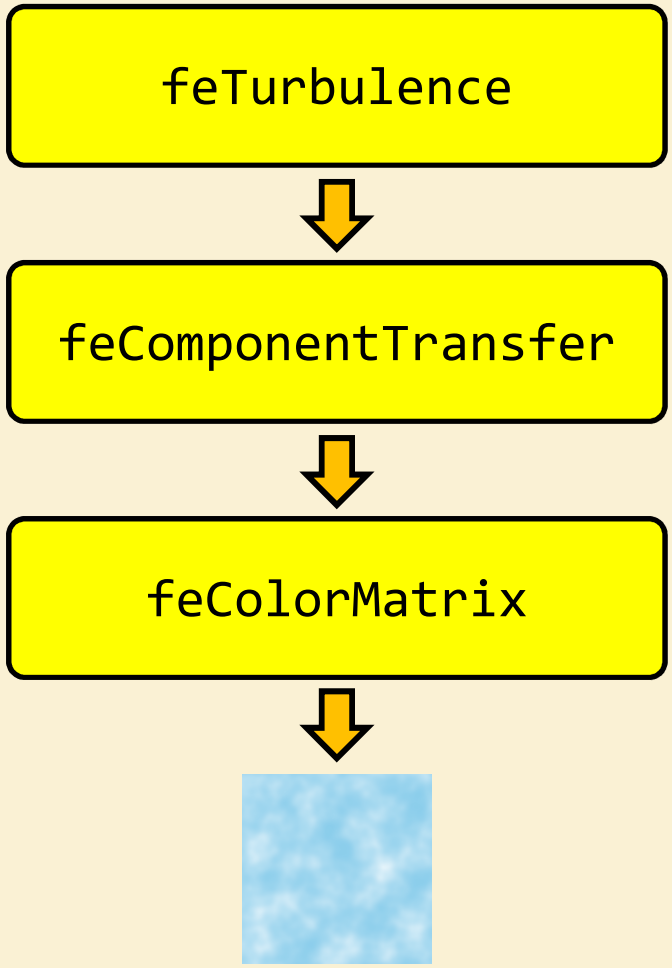
- The two grass layers are two rectangles using the same grass filter
- Two layers of grass are used in the display for the same texture because the L-system tree will be put in between the two layers later, so that the bottom of the tree trunk will be hidden by one of the grass layers



The Sky Texture

- The sky texture is a simple Perlin noise texture that we have discussed in the previous lectures
- You will need at least three filter effects to create the sky texture:
  - `feTurbulence`
  - `feComponentTransfer`
  - `feColorMatrix`

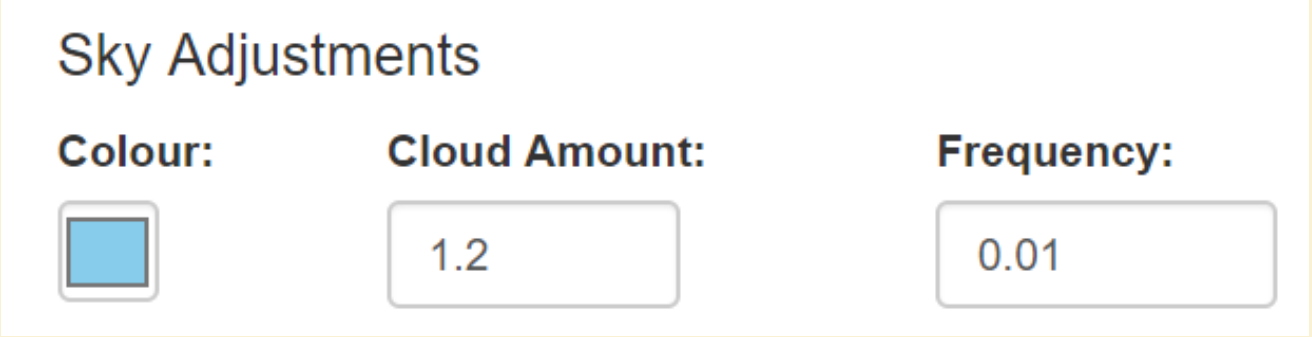
Filter Effects for Sky Texture



Note that you may use other filter effects and a different filter effect order

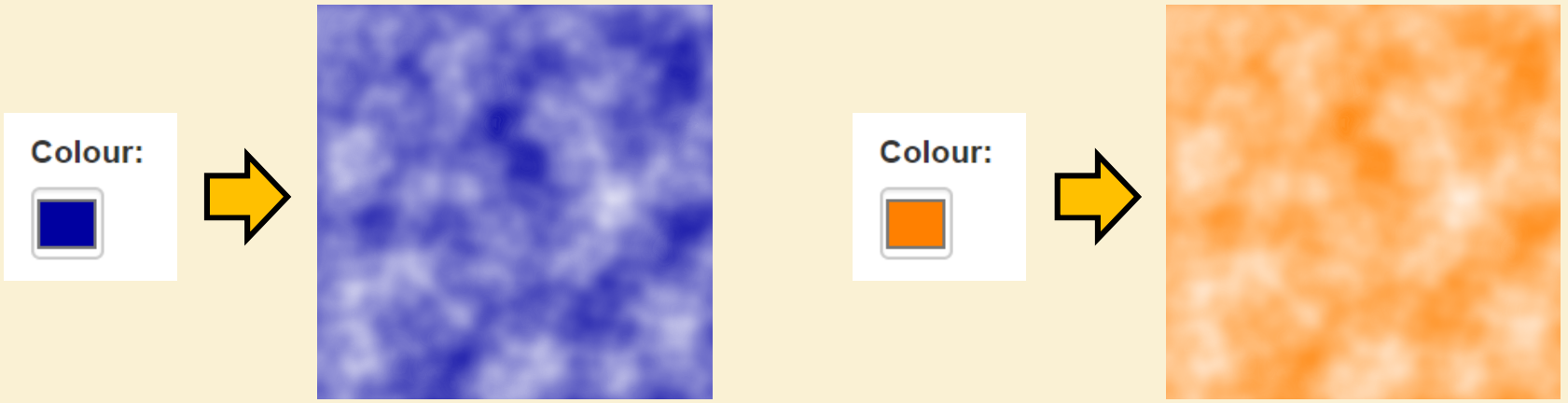
The Controls for the Sky Texture

- You can control the sky texture using the following parameters:



Controlling the Sky Colour

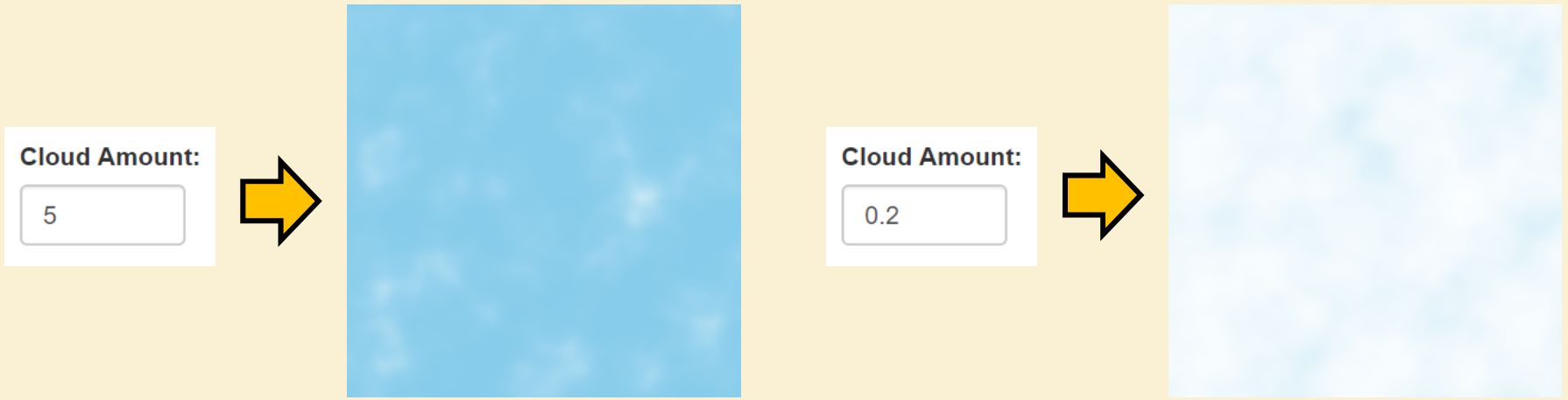
- You can change the sky colour to any other colour using the colour control
- Here are a couple of examples:



- In the system, we assume the clouds are always in white colour

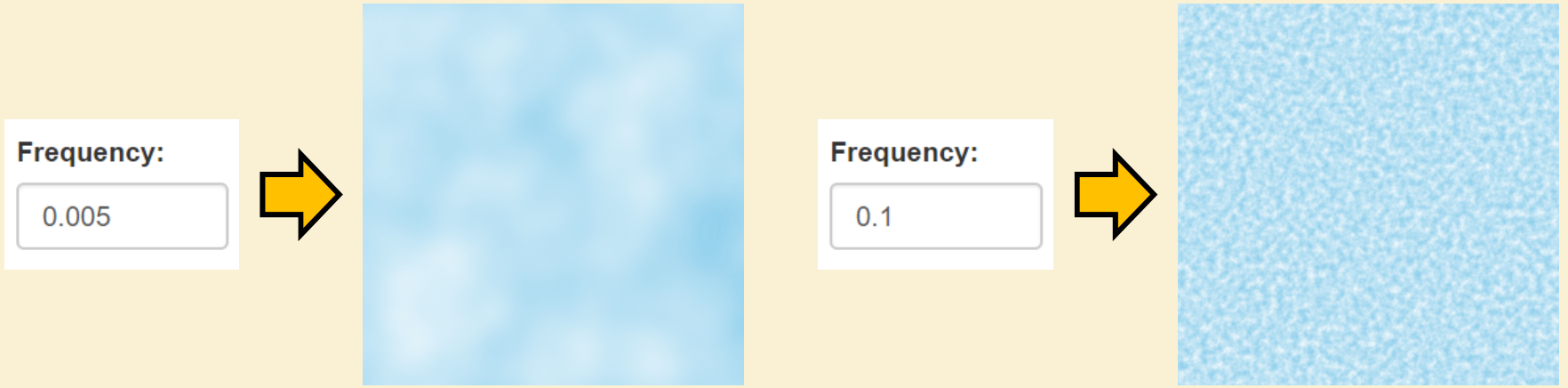
Changing the Amount of Clouds

- You can change the amount of clouds by adjusting the ‘Cloud Amount’ parameter
- The parameter is a number which maps to the exponent of the gamma transfer
- Here are a couple of examples:



Setting the Cloud Frequency

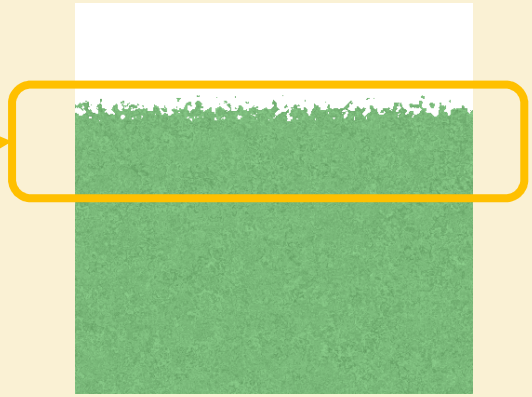
- The cloud frequency corresponds to the frequency of the changing clouds, which is mapped to the frequency of the Perlin noise data
- Lower frequencies usually work better for clouds:



- For the assignment, you can assume the number of octaves for `feTurbulence` is always 3

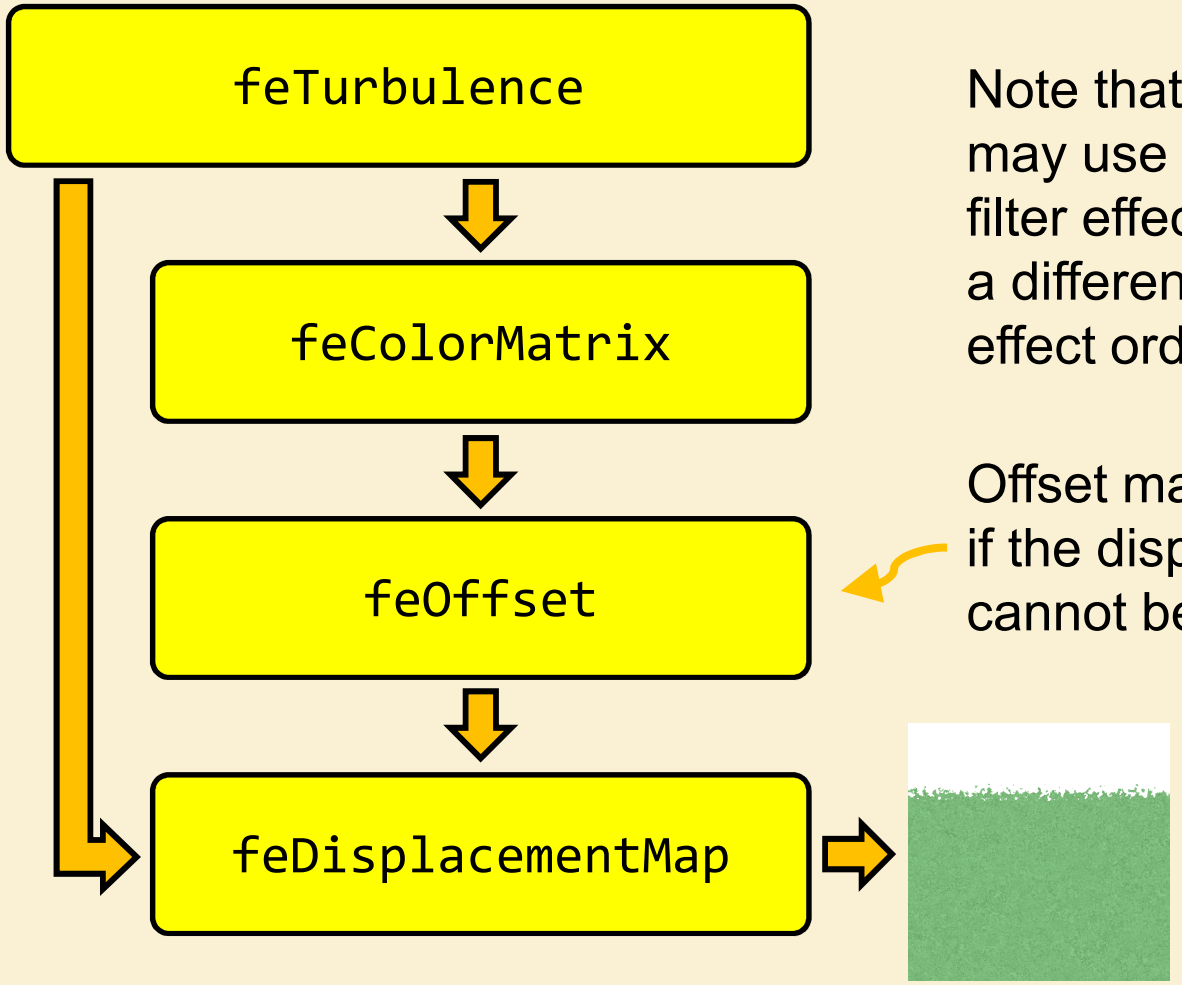
The Grass Texture

- The grass texture is a variation of the sky texture, plus a rough edge at the top of the grass area



- To make the grass texture, you will need a slightly different set of filter effects:
  - `feTurbulence`
  - `feColorMatrix`
  - `feOffset`
  - `feDisplacementMap`

Filter Effects for Grass Texture

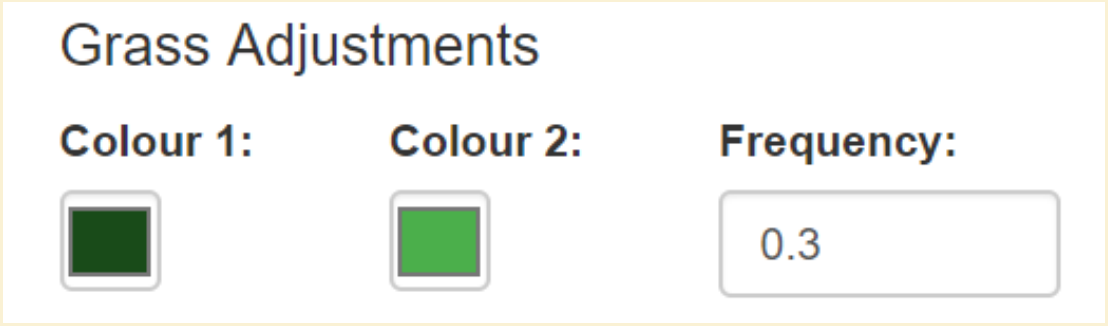


Note that you may use other filter effects and a different filter effect order

Offset may be needed if the displacement cannot be shown

The Controls for the Grass Texture

- You can control the grass texture using the following parameters:



Changing the Grass Colours

- The colour mapping of the grass texture has two colours
- You can change the two colours by adjusting the colour controls
- For example, the mapping can be changed to, yellow to red:

Setting the Grass Frequency

- Similar to the cloud parameter, the grass frequency is mapped to the frequency of the Perlin noise data
- For grass, higher frequencies usually work better:

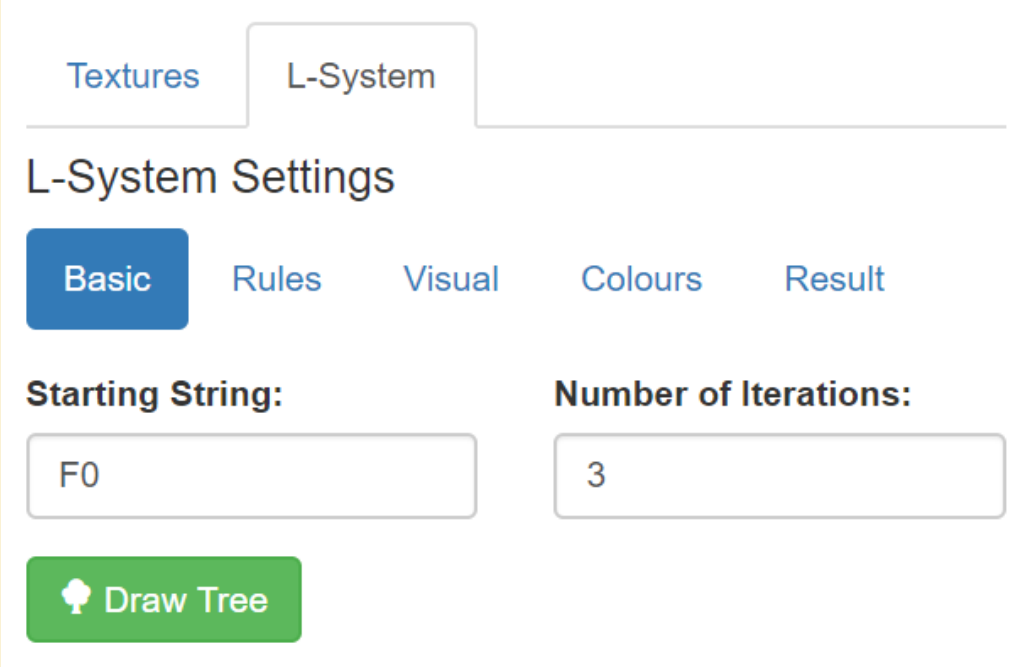
- Since the Perlin noise data is used in both the grass texture and the displacement, adjusting the frequency will affect them both

Part 2 – An Extended L-System

- The starting code gives you a basic L-system that can create some simple L-system curves
- Part 2 of the assignment aims at extending the L-system to draw trees by:
  1. Adding a stack and the corresponding [and] symbols to the L-system
  2. Including the use of colours for different symbols
  3. Adjusting the length and width of the lines based on a depth parameter

The Control Area for Part 2

- Here is the control area for part 2:
- We will look at the tabs in the next few slides

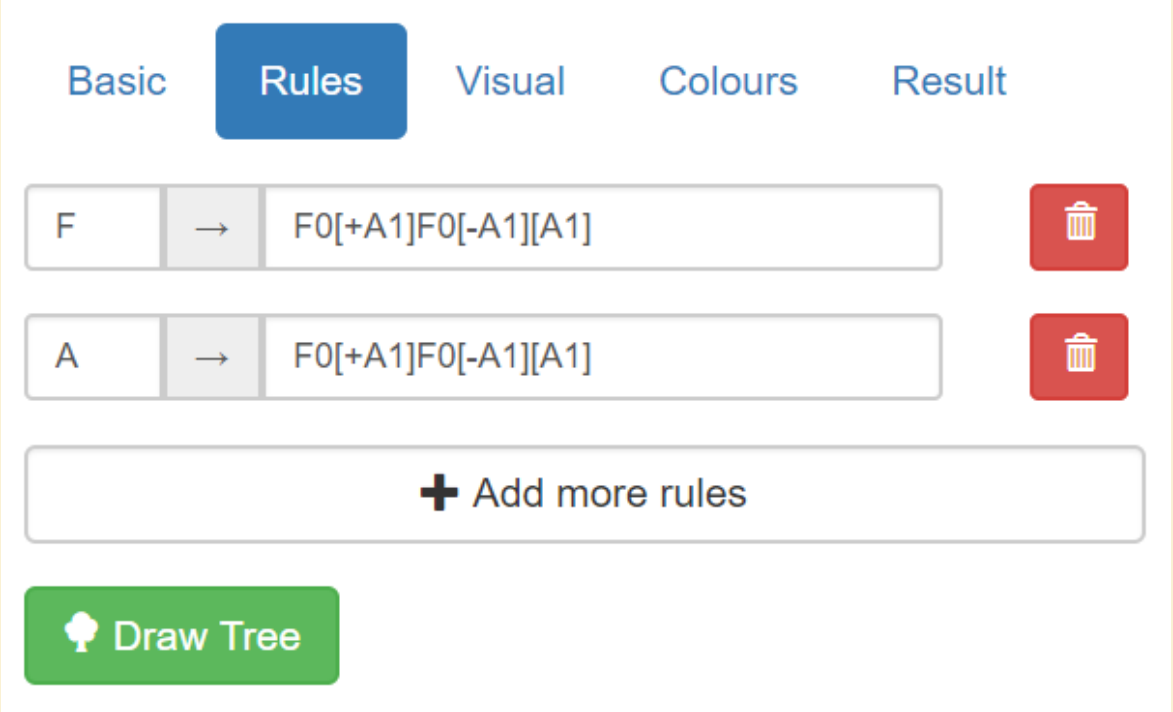


The Basic Tab

- The basic tab defines the starting string of the L-system and the number of iterations to run for the system

The Rules Tab

- In the rules tab, you can set as many as 5 replacement rules for the L-system

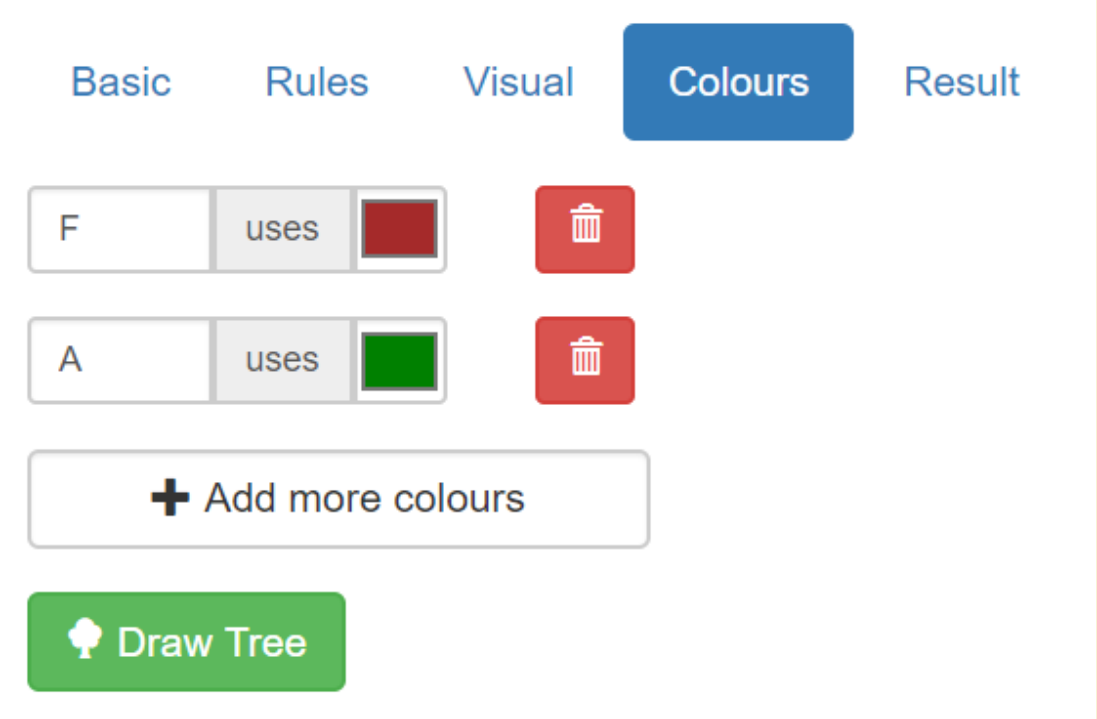


The Visual Tab

- The visual tab contains some visual parameters of the L-system, such as the length and width of the lines and the angle to turn each time
- The length and width ratios are used for adjusting the length and width of some tree branches later

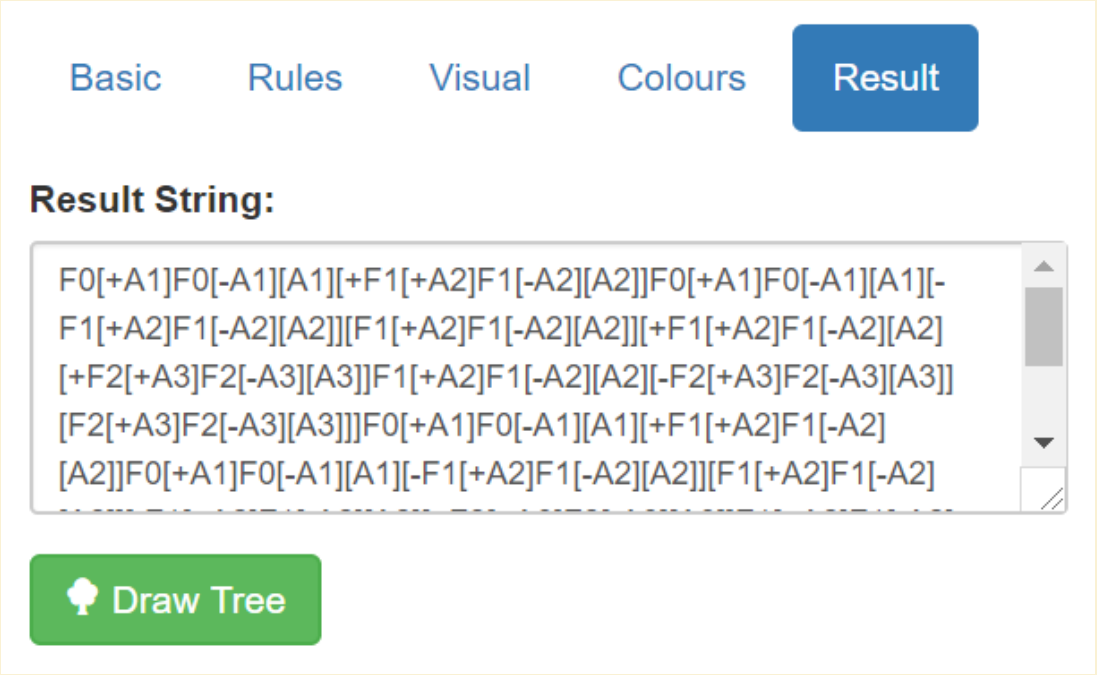
The Colours Tab

- You define the colours to be used for some symbols in the L-system in the colours tab



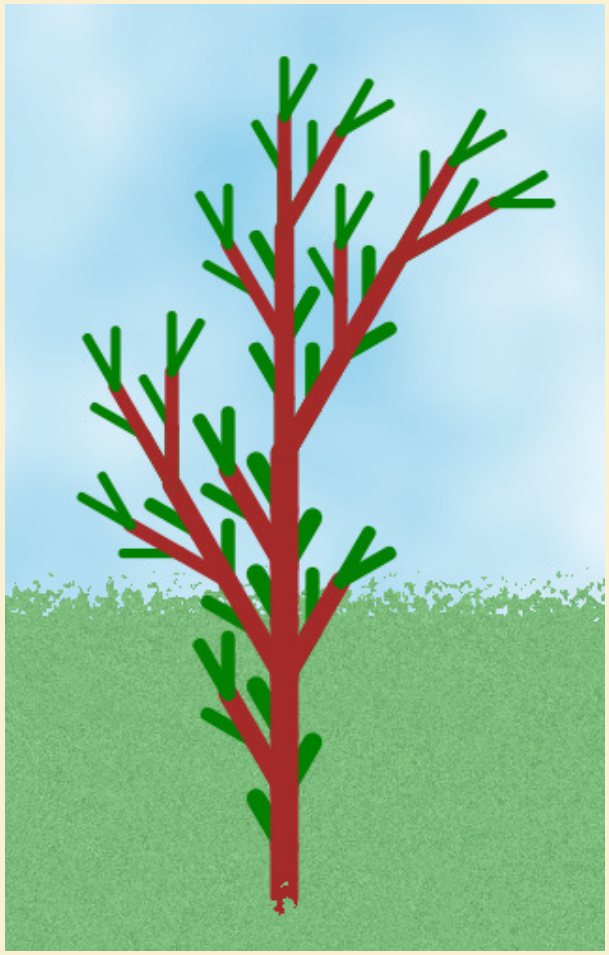
The Result Tab

- The result tab has a text area which shows the last L-system string generated by the L-system
- It is very useful for debugging purpose



The L-System Output

- Using the L-system parameters shown in the previous slides, the tree shown on the right can be generated by the extended L-system:



Downloading the Starting System

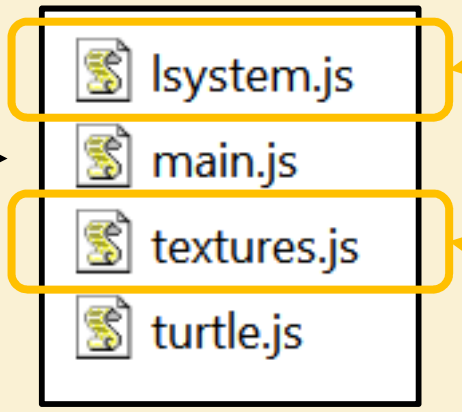
- You can download the starting system from the course canvas website
- The files of the system are put inside a zip file
- You need to extract the files from the zip file into an appropriate folder and start the system by opening `index.html` in a browser, preferably in Chrome

File Structure

Starting HTML file

`index.html`

`js`



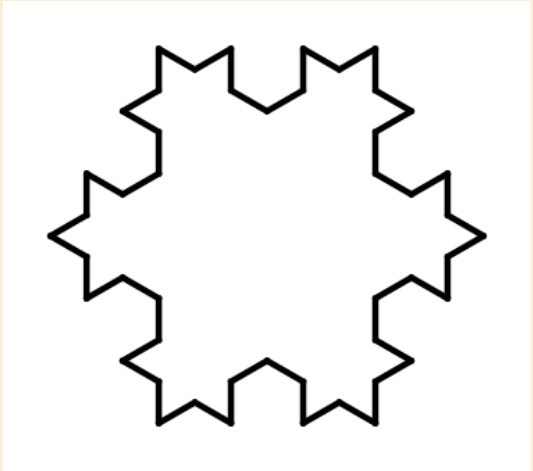
You change this JavaScript file for part 2 of the assignment

You change this JavaScript file for part 1 of the assignment

JavaScript files, all your code will be added here

The Given L-System

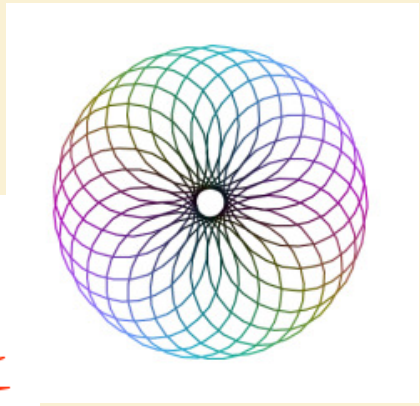
- In the starting system, if you draw the L-system using the ‘Draw Tree’ button, the parameters will give you a Koch snowflake:

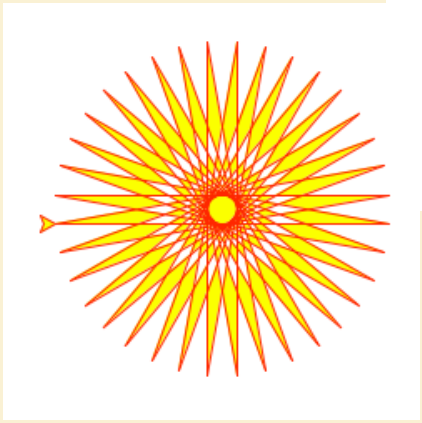


- If you like, you can adjust the parameters to draw different L-system curves
- However, not all parameters, such as colours, are supported in the starting system

The Turtle Graphics System

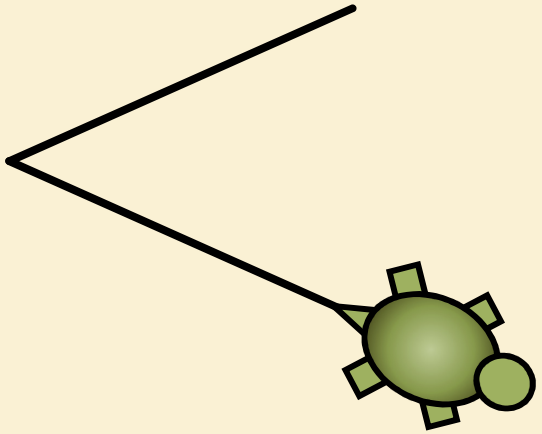
- The `turtle.js` file contains a JavaScript implementation of the turtle graphics system
- The turtle graphics system is a simple drawing system that can create interesting line graphics
- The assignment uses this turtle graphics system to draw the L-system images





Using the Turtle

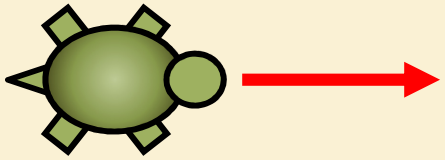
- The turtle graphics system relies on a ‘turtle’ to draw things in a 2D drawing area
- The turtle holds a pen so it leaves lines behind when it moves in the drawing area



- To draw things using a program, you write a series of commands to move the turtle around
- These turtle graphics commands are mainly for moving the turtle as well as changing the properties of the pen

Position and Heading

- The turtle has a position and heading



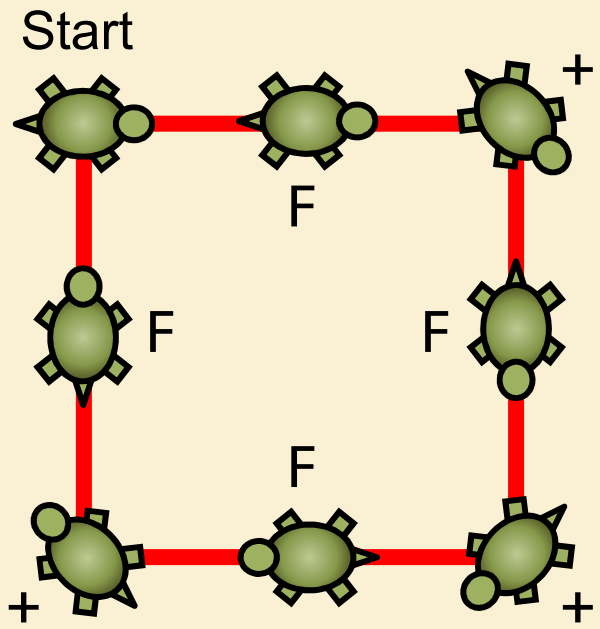
- The position is the current x and y position of the turtle in the drawing area and the heading is the direction the turtle is looking at, which is an angle from 0 to 360 degree
  - A angle of 0 means looking to the right
- You can ask the turtle to move based on its position and heading; or move it to an absolute position in the drawing area

Drawing an L-System String

- For example, given the following L-system string:

F+F+F+F

- You can ask the turtle to draw the string by asking the turtle to move forward when you see the symbol ‘F’ and turn right when you see the symbol ‘+’
- The turtle will produce a square from the L-system string



The Turtle Graphics Commands 1/2

- In these two slides, we list the supported turtle graphics commands in the system
- Here are the commands for moving:
  - `turtle.goto(`*x*`, `*y*`)`
  - `turtle.home()`

  } Go to an absolute position, where home means (0, 0)

  - `turtle.forward(`*length*`)`
  - `turtle.backward(`*length*`)`

  } Move forward or backward by a certain distance

  - `turtle.setHeading(`*heading*`)`

  } Change the heading of the turtle

  - `turtle.left(`*angle*`)`
  - `turtle.right(`*angle*`)`

  } Turn left or right

The Turtle Graphics Commands 2/2

- Here are the commands for setting the pen properties:
  - `turtle.color(`*color*`)` } Change the colour (=line colour) of the pen
  - `turtle.width(`*width*`)` } Change the width (=line thickness) of the pen
  - `turtle.up()`
  - `turtle.down()` } Lift up the pen so no line is drawn or put down the pen so lines are drawn
- Here are some other useful commands:
  - `turtle.pos()`
  - `turtle.heading()` } Get the current position, as an array of [x, y] and the heading of the turtle
  - `turtle.reset()` } Reset and clear the turtle area

An Example Turtle Graphics Program

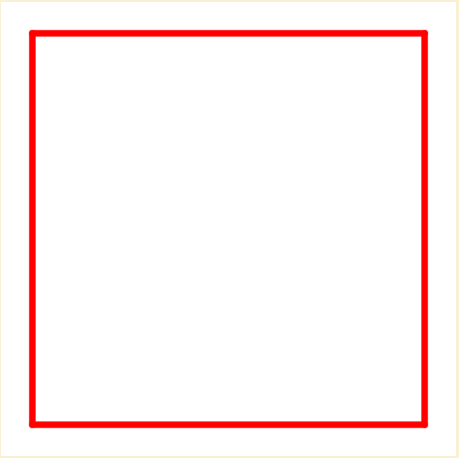
```
var turtle = new Turtle($("#drawing_area"));

turtle.up();
turtle.goto(50, 50);
turtle.down();

turtle.color("red");
turtle.width(5);

turtle.forward(300);
turtle.right(90);
turtle.forward(300);
turtle.right(90);
turtle.forward(300);
turtle.right(90);
turtle.forward(300);
```

Move to the top-left corner of the square

Change the line to a red and thick line

Ask the turtle to move around the square

- This program draws a square in an SVG group called ‘drawing_area’



The Drawing Area in the Assignment

- You have seen the SVG content from the part 1 discussion

```
<rect x="0" y="0" width="500" height="500"
      filter="url(#sky)"/>
<rect x="0" y="300" width="500" height="200"
      filter="url(#grass)"/>

<g id="tree"/>

<rect x="0" y="400" width="500" height="100"
      filter="url(#grass)"/>
```

This is the group where the turtle will draw on

- The turtle will draw inside the ‘tree’ group

The Given L-System Code

- You will need to read thoroughly the given L-system code before working on the extensions
- The code is inside the file `lsystem.js`
- The code starts from the `drawTree()` function, which runs when you click on the ‘Draw Tree’ button

The drawTree() Function

- The `drawTree()` function mainly does three things:
  1. Reading the L-system parameters from the HTML components
     (Not all of them are used in the starting system)
  2. Running the `runLSystem()` function to generate the L-system string
  3. Running the `drawLSystem()` function to draw the L-system image from the string

Your Tasks

- From the `lsystem.js` file, you will need to complete the following tasks:

  Task 1 – Using a Stack

  Task 2 – Changing Symbol Colours

  Task 3 – Adding Depth

Task 1 – Using a Stack

- To use a stack, first, you create a stack structure before you draw the L-system
  - In JavaScript, a stack can be implemented by a simple array, as shown on the next slide
- Then, when you see the [and] symbols, you use the stack to remember and restore the position and heading of the turtle respectively

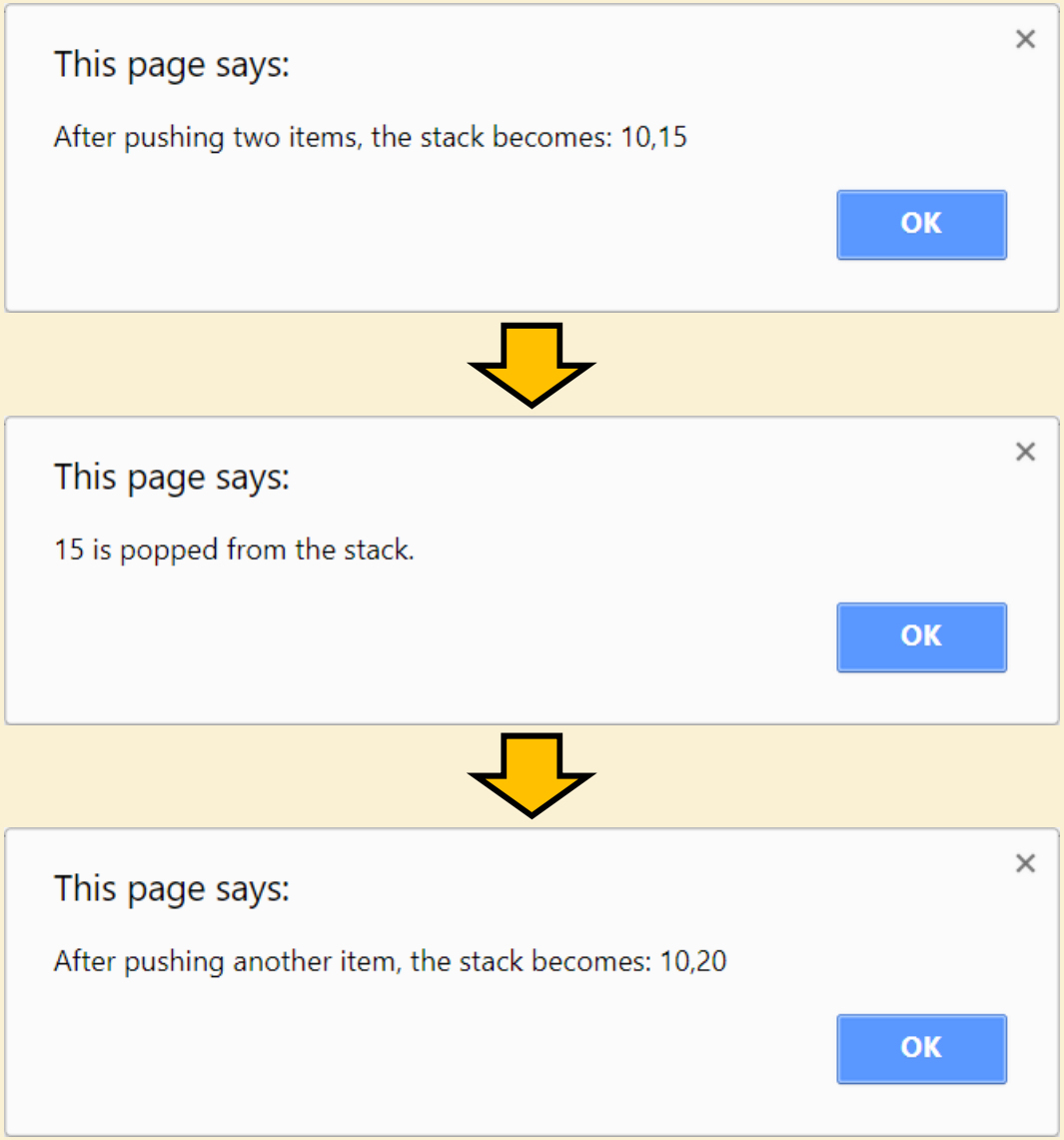
Using an Array as a Stack 1/2

- Here is some JavaScript that uses an array as a stack:

```
var stack = [];

stack.push(10);
stack.push(15);
alert("After pushing two items, the stack becomes: " + stack);

var item = stack.pop();
alert(item + " is popped from the stack.");

stack.push(20);
alert("After pushing another item, the stack becomes: "
      + stack);
```

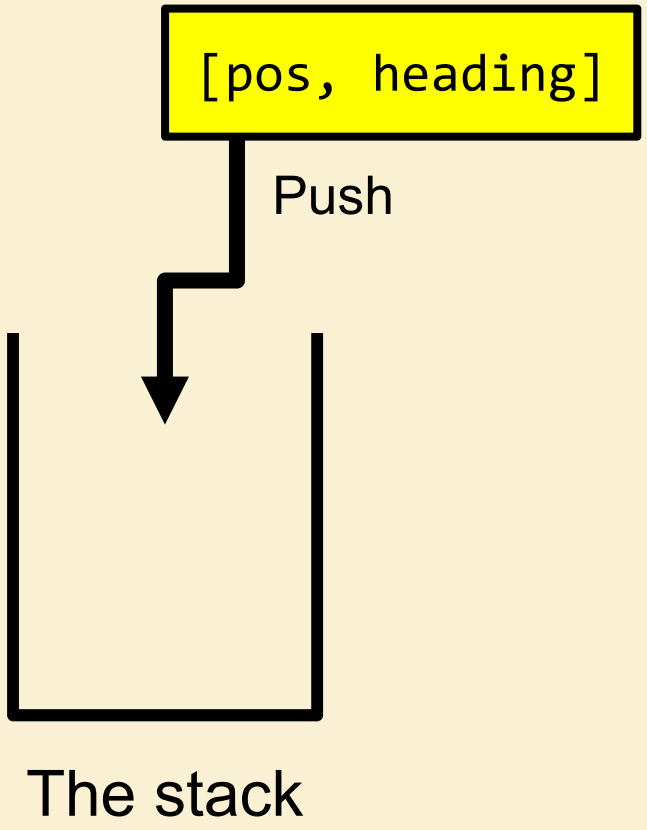
Using an Array as a Stack 2/2

- Here is the output of the previous example:



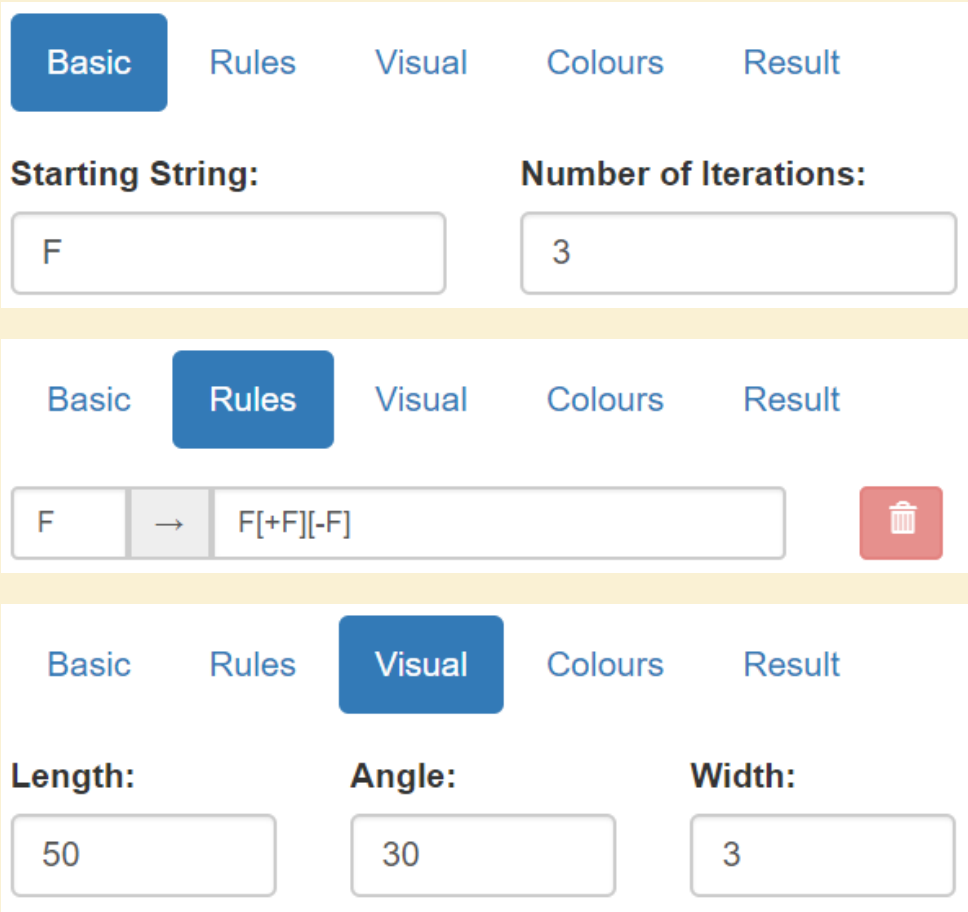
Putting Information onto the Stack

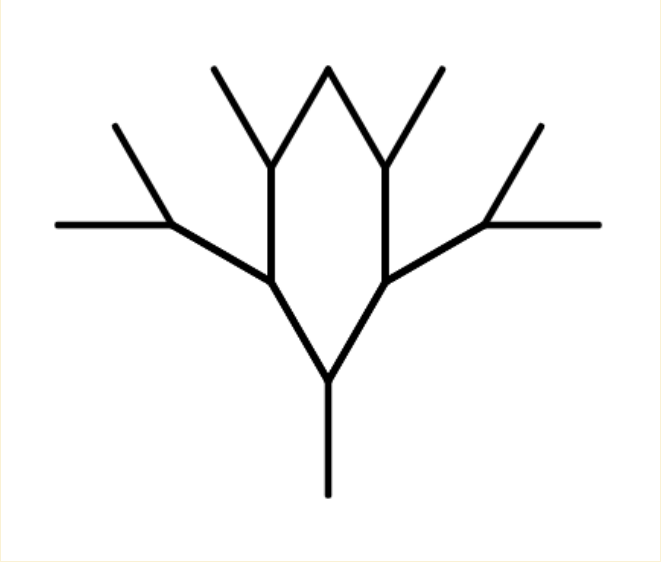
- The stack stores both the turtle position and heading as a single item
- You need to find a way to push both values as a single item onto the stack
- You can do that by putting them into an array before pushing



After Finishing Task 1

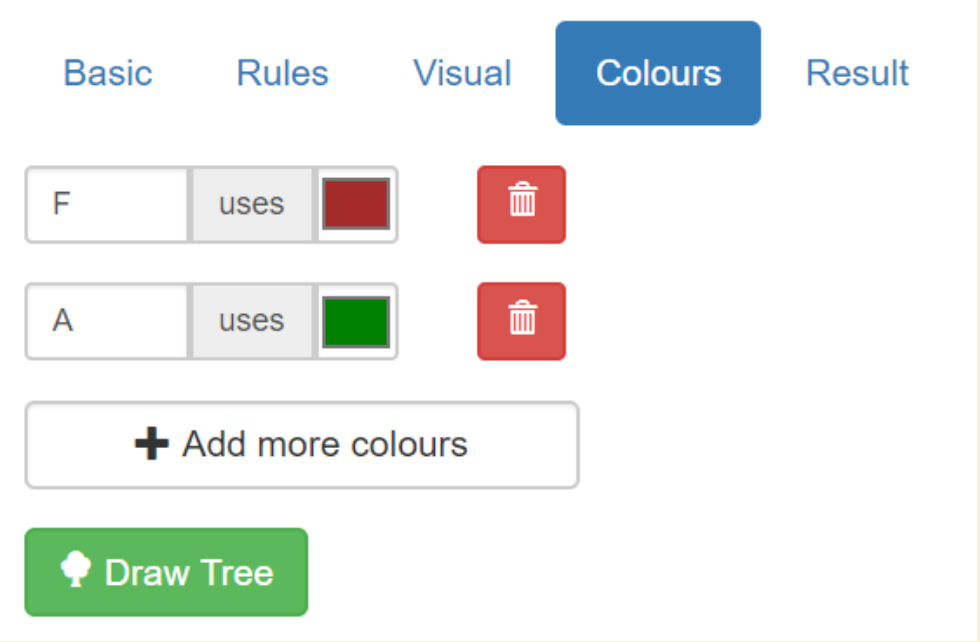
- After you finish this task, you should be able to show a tree, for example, like this:





Task 2 – Changing Symbol Colours

- After task 1, the tree is completely black
- You will add some colours to the tree by simply assigning different colours to different symbols
- The colours are specified in the colours tab:
  - ‘F’ is red and ‘A’ is green in the example



The Colours in the Code

- The colour mapping has been stored in the `colors` object in the code, which is initialised like this:

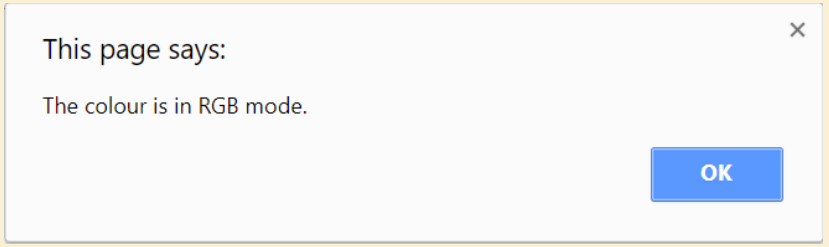
```
var colors = {};
```

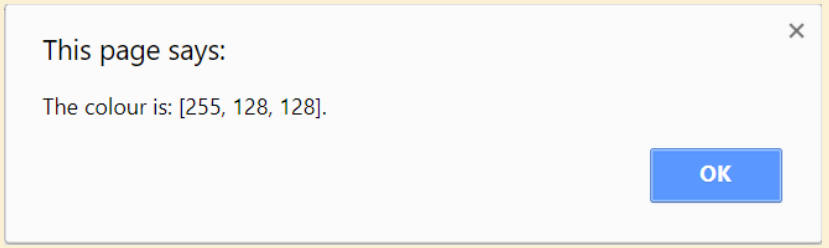
- A JavaScript object can also be used as an associative array
- The `colors` object in the code has been used like that

Using Objects as Assoc. Arrays 1/2

- Here is an example that builds an associative array and then works with the content of the array:

```
var color = {};

color["red"]   = 255;
color["green"] = 128;
color["blue"]  = 128;
```





To next page

Using Objects as Assoc. Arrays 2/2

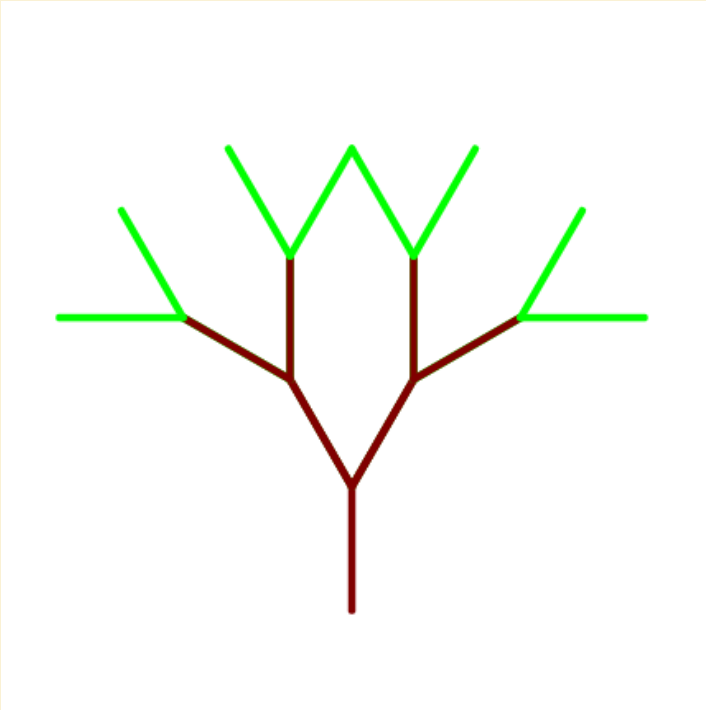
From previous page

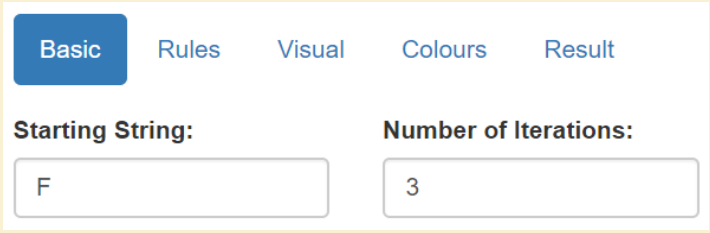
```
if ("alpha" in color) {
    alert("The colour is in RGBA mode.");
    alert("The colour is: [" +
          color["red"]  + ", " + color["green"] + ", " +
          color["blue"] + ", " + color["alpha"] + "].");
}
else {
   alert("The colour is in RGB mode.");
   alert("The colour is: [" + color["red"] + ", " +
         color["green"] + ", " + color["blue"] + "].");
}
```

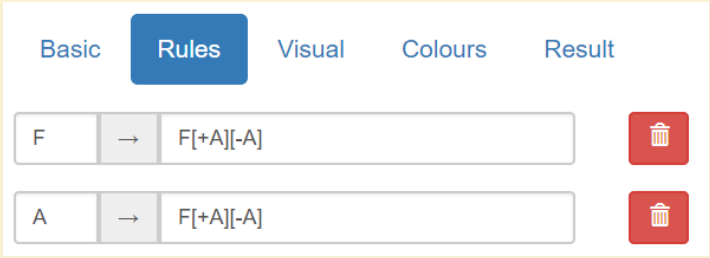
Only this part is run because
"alpha" is not in the object/array

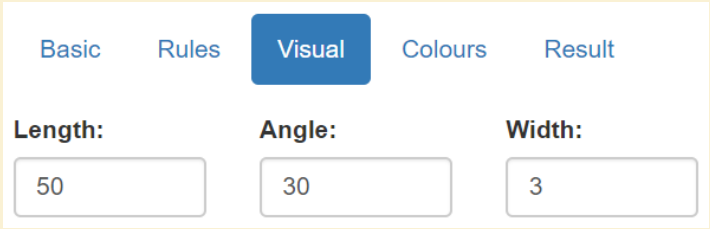
After Finishing Task 2

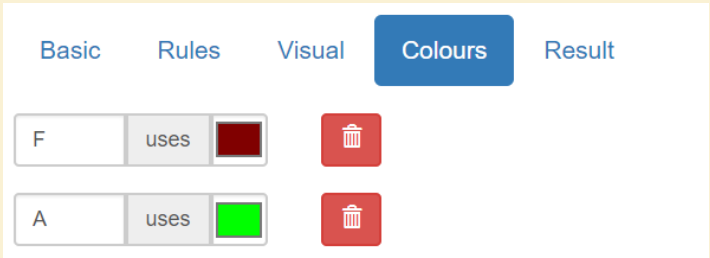
- After you finish this task, you should be able to show a coloured tree
- Here is an example:





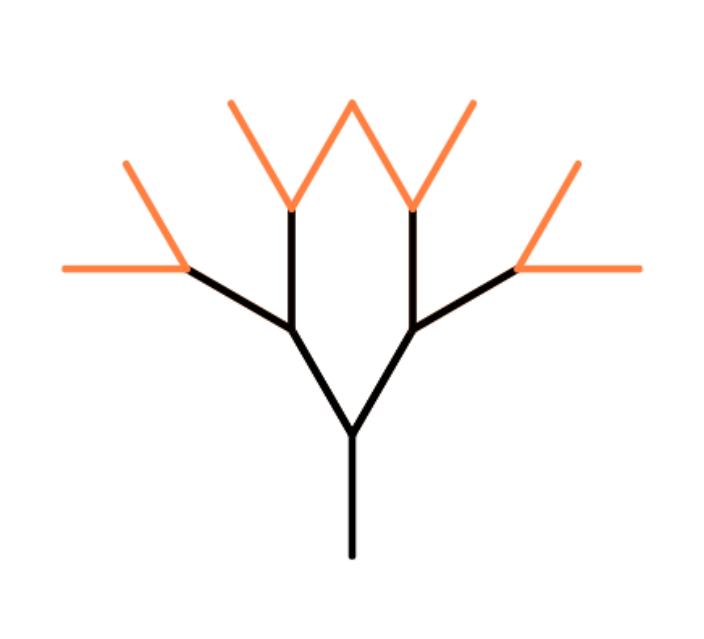


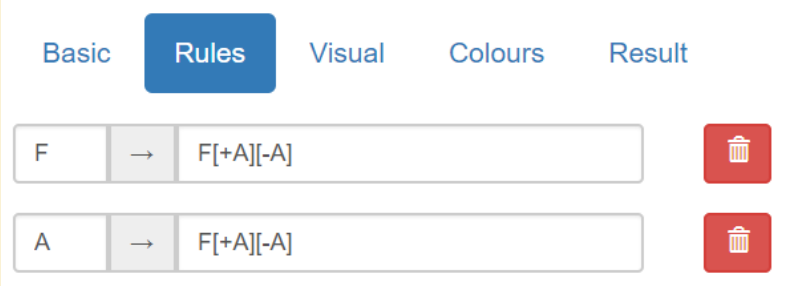


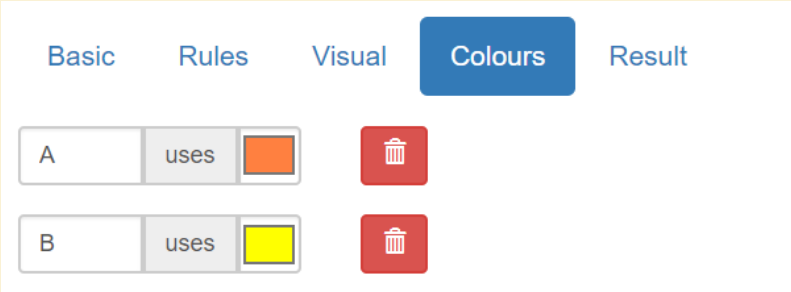


The Default Colour

- If a symbol has not been mapped to any colour, the default colour will be black







Task 3 – Adding Depth

- The final task is to add a ‘depth’ parameter to the system
- The parameter is written next to a symbol
  - For example, in the following L-system string:

```
F1[+A2][-A2]
```

    the depth value of ‘F’ is 1 and the depth value of both ‘A’ is 2
- The depth values are used for adjusting the length and width of the symbols later

Adjusting the Depth Value

- A symbol may have an associated depth value such as `F1` and `A2` in the previous example
- If such a symbol is replaced, its depth value will also be put inside the replacement string at appropriate places
- The depth value to be put in the replacement string is adjusted by adding on top a given increment specified in the replacement rule
- An example is shown on the next slide

A Replacement Rule with Depth

- For example, in the assignment, a replacement rule can be written like this:

$$\texttt{F} \rightarrow \texttt{F0A1B}$$

- This is equivalent to the following rule in a parametric L-system:

$$\texttt{F}(d) \rightarrow \texttt{F}(d+0)\texttt{+A}(d+1)\texttt{B}$$

- If the incoming symbol has a depth d, the value of d will be increased for the two symbols `F` and `A` in the replacement rule

A Replacement Example

- Let's apply the replacement rule from the previous slide to this L-system string:

```
F1[+A2][-A2]
```

- The resulting L-system string will become:

```
F1A2B[+A2][-A2]
```

This replacement is coming from the replacement rule, where: `F(1) → F(1+0)A(1+1)B`

Another Example

- If the L-system string is again `F1[+A2][-A2]` and the replacement rule is:

  `A → A4F0`

  the resulting L-system string will become:

  `F1[+A6F2][-A6F2]`

1. `A2` is replaced by `A4F0`
2. Its depth value 2 has been put and added in the location of 4 and 0 in the replacement, to become 2 + 4 = 6 and 2 + 0 = 2 respectively

A Complete Example

- If the L-system has the following inputs:
  - Starting string: `F0[+A1][-A1]`
  - Rule: `A → F0[+A1][-A1]`
  - Number of iterations: 1
- The resulting L-system string will be:

```
F0[+F1[+A2][-A2]][-F1[+A2][-A2]]
```

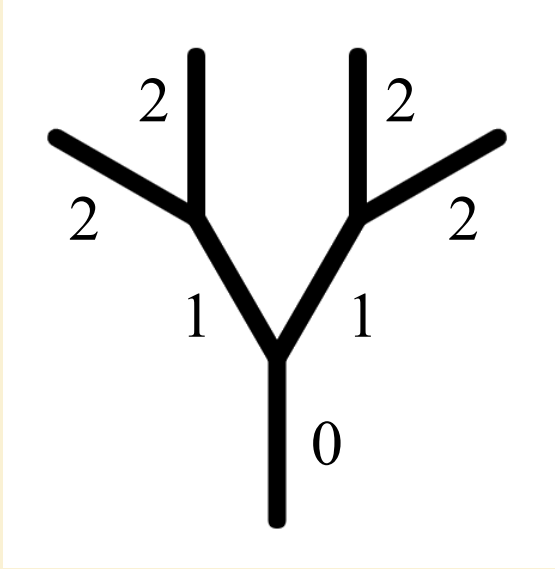
- The depth value can be used to indicate the number of generation of branches in a tree

Generation of Branches

- The tree shown below is drawn from the L-system string:

```
F0[+F1[+A2][-A2]][-F1[+A2][-A2]]
```

- The generation number is also shown next to each branch



Extracting the Depth in the Code

- In the starting system, a function:

  ```
  getNumberFromString(string, index)
  ```

  has been given to you so that you can extract a number from an L-system string
- You can use the function to read the number at a given location
- If there is no number at the given location, the number returned will become `NaN`, which is a JavaScript value meaning ***N**ot **a N**umber*
- The `NaN` value can be detected by `isNaN()`

Using the Function

- For example, if the given string is “`A10B5C6`” and the index is 1, the function will return the following JavaScript object:

```
{ number: 10, next: 3 }
```

- Or, if the given string is “`F[+A][-A]`”and the index is 7, the function will return the following JavaScript object:

```
{ number: NaN, next: 7 }
```

Using the Depth Values

- The depth values can be used to adjust the length and width of the branches
- The idea is that the younger a branch is (i.e. having a bigger depth), the thinner and shorter it is
- Using the length and width ratios, the reduction of length and width can be applied to each branch based on its depth value

Adjusting the Length and Width

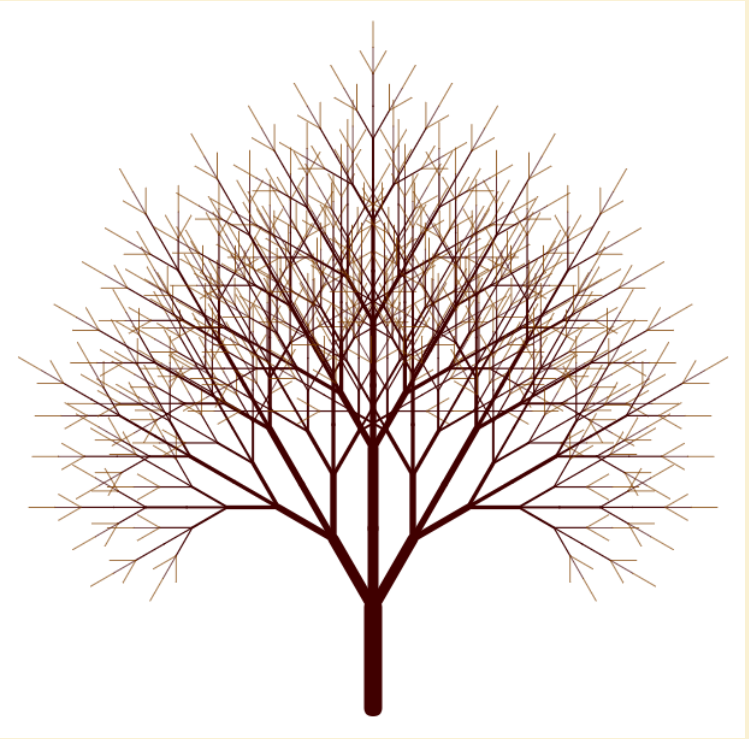
- For example, if the length is 10 initially and the length ratio is 0.8, then
  - the length of `F0` will be 10
  - the length of `F1` will be 10 * 0.8
  - the length of `F2` will be 10 * $(0.8)^2$
  - and so on
- In general, the length of `F`n will be:

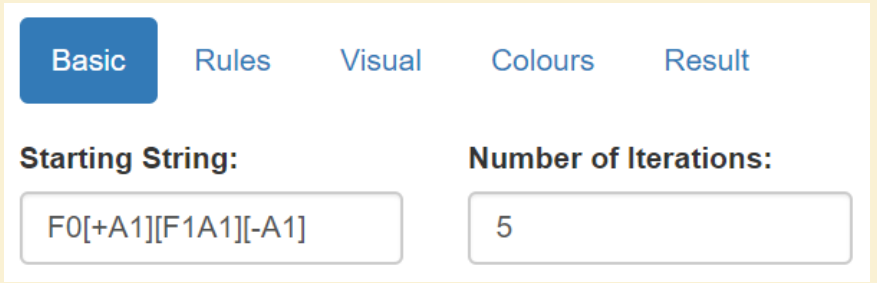
$$initial\ length \times (length\ ratio)^n$$

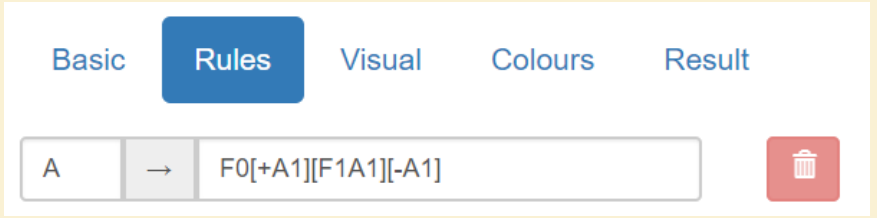
- The same can be applied to the width value

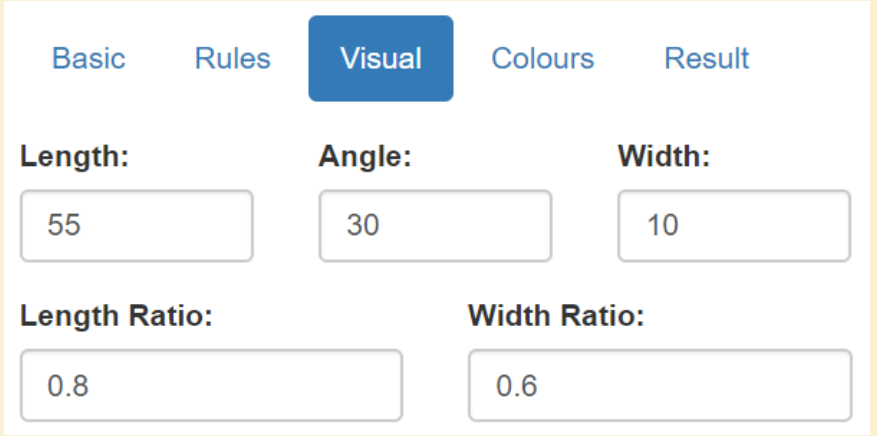
After Finishing Task 3

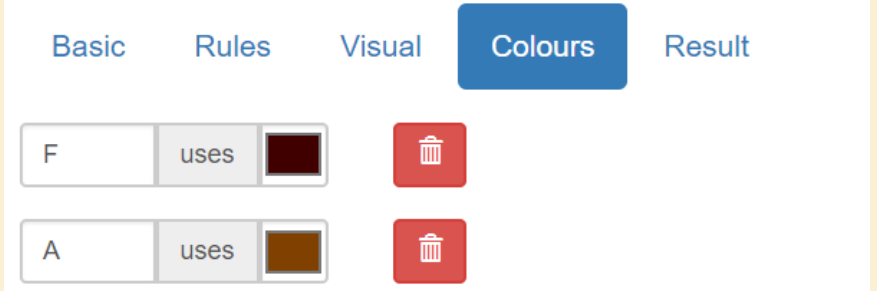
- After finishing all tasks, you can create a tree which looks like this:





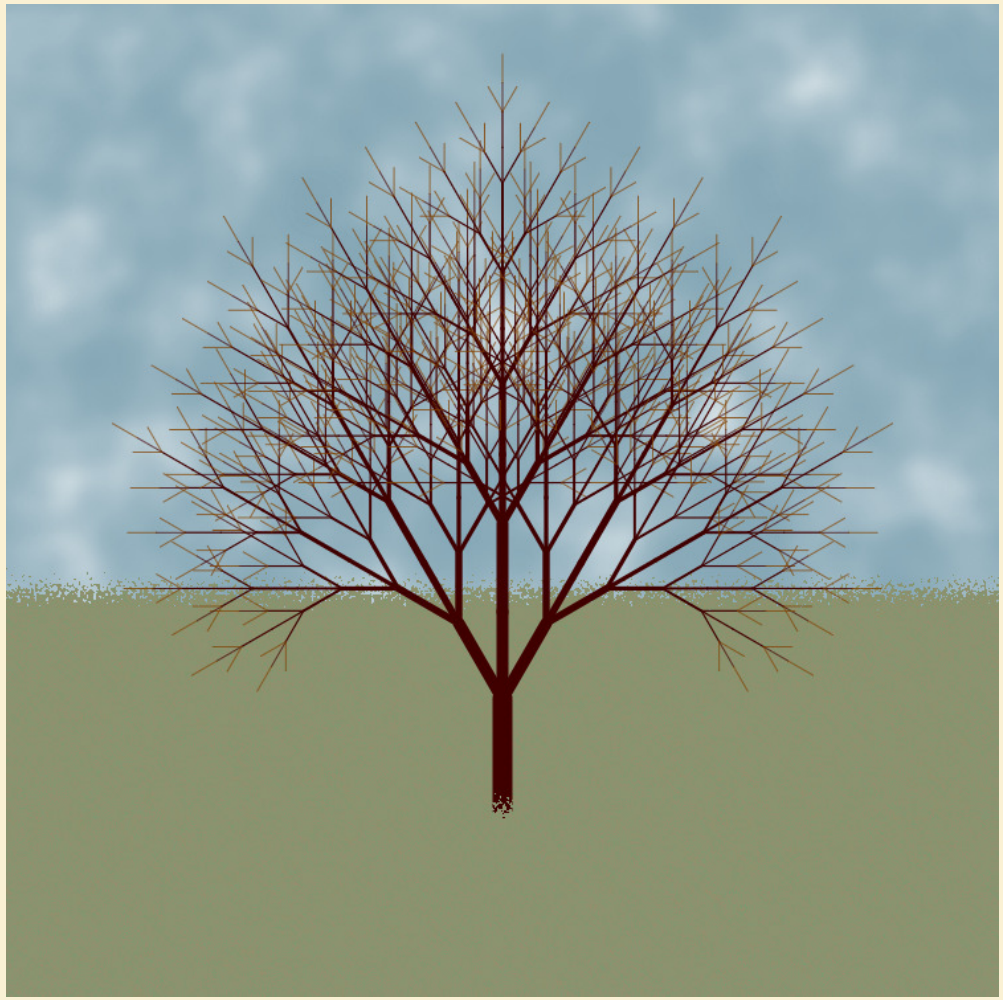






Combining Part 1 and Part 2

- Using both the results of part 1 and part 2, the tree image can become:



Marking Scheme 1/2

- Total marks is 100
  - The sky texture
    - A correct sky texture can be generated 10%
    - The sky colour can be changed correctly 5%
    - The cloud amount can be changed correctly 5%
    - The cloud frequency can be changed correctly 5%
  - The grass texture
    - A correct grass texture can be generated 10%
    - The two layers of grass texture are properly done 5%
    - The grass colours can be changed correctly 5%
    - The grass frequency can be changed correctly 5%

Marking Scheme 2/2

- Total marks is 100
  - Using L-system stack
    - L-system images are drawn correctly when the stack symbols are used in the L-system string 10%
  - Using L-system colour
    - Correct colours are used based on the colour mapping 5%
    - The default colour is black 5%
  - Using the depth parameter
    - Depth values are put at appropriate places using replacement rules containing depth increments 10%
    - Depth values are correctly increased in the result 10%
    - Length is adjusted correctly using the depth values 5%
    - Width is adjusted correctly using the depth values 5%

Submission

- The deadline of the assignment is:

8pm, Saturday, 24 Nov 2018

- To submit your assignment:
  - You need to put **everything** (HTML file and JavaScript files) into a zip file called `<your ITSC account>_a2.zip`
  - For example, if your ITSC account is *johnc*, you will put your files into `johnc_a2.zip`
  - You can then submit the zip file through canvas