

Programming with Edgy

Charlotte Wilson, Steven Bird

Programming with Edgy

Charlotte Wilson, Steven Bird

Generated by [Alexandria](https://www.alexandriarepository.org) (https://www.alexandriarepository.org) on November 26, 2018 at 10:26 am AEDT

Contents

Title	i
Copyright	ii
1 Tutorials	1
1.1 <i>Getting Started with Edgy: Creating A Network</i>	2
1.2 <i>A First Program: Using Blocks to Create a Social Network</i>	6
1.3 <i>Generating a Social Network: Processing Lists and Graphs</i>	12
1.4 <i>Exploring your Social Network</i>	17
1.5 <i>Who is friends with who?</i>	21
1.6 <i>Making Friend Recommendations</i>	26
1.7 <i>Visualisations for Social Networks</i>	31
1.8 <i>Visualising the Spread of News in a Social Network</i>	36
1.9 <i>Appendix: From Edgy to Python</i>	42

1 Tutorials

- [1.1 Getting Started with Edgy: Creating A Network](#)
 - [1.2 A First Program: Using Blocks to Create a Social Network](#)
 - [1.3 Generating a Social Network: Processing Lists and Graphs](#)
 - [1.4 Exploring your Social Network](#)
 - [1.5 Who is friends with who?](#)
 - [1.6 Making Friend Recommendations](#)
 - [1.7 Visualisations for Social Networks](#)
 - [1.8 Visualising the Spread of News in a Social Network](#)
 - [1.9 Appendix: From Edgy to Python](#)
-

1.1

Getting Started with Edgy: Creating A Network

This lesson provides an introduction to Edgy. Edgy is a visual programming language and algorithm design environment. In Edgy, you can create programs by dragging blocks of code together. In this lesson, you'll create a first project and discover how to use the Edgy user interface to create, load and save a graph or network. You will create nodes and edges in a graph to represent a social network.

Tutorial - Creating a Network with Edgy

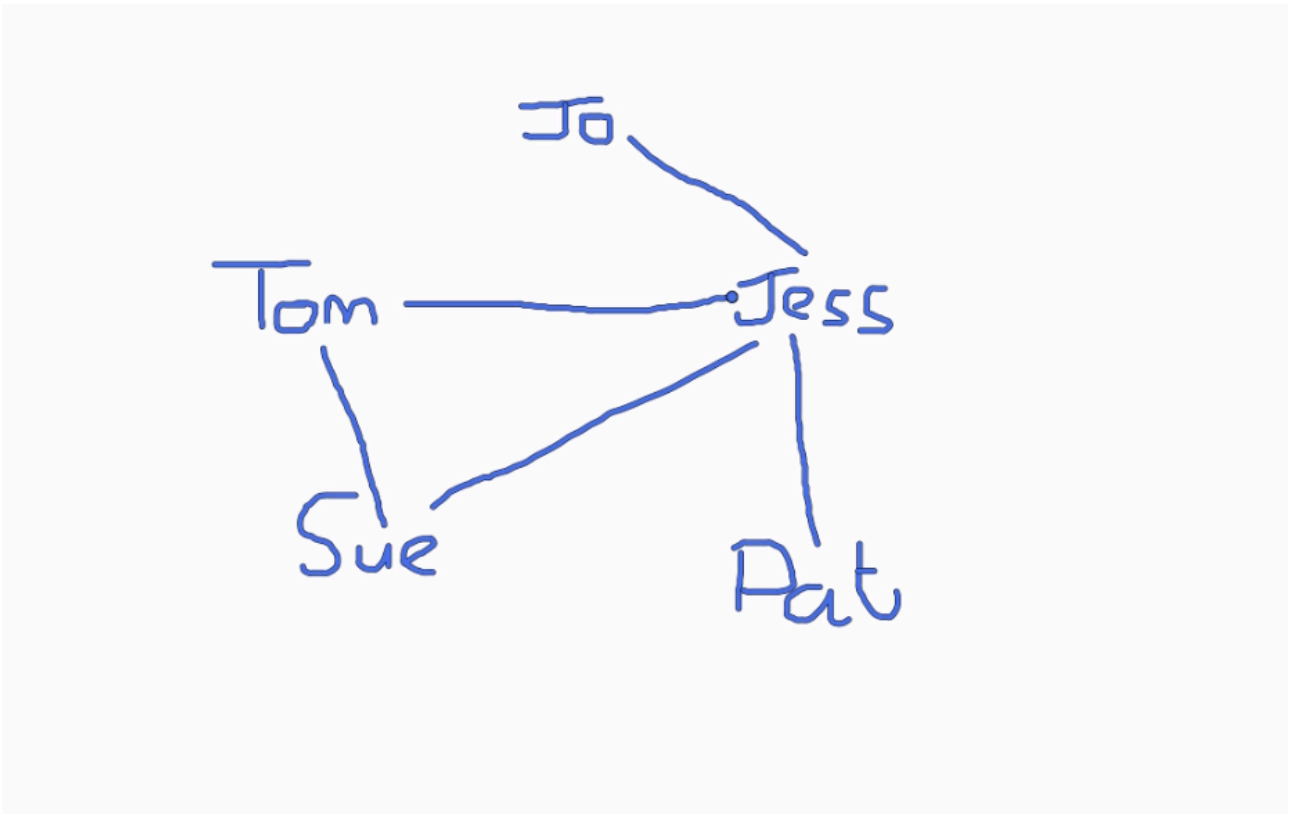


(https://www.alexandriarepository.org/wp-content/uploads/20161216110135/Edgy_intro_v2_strm.mp4)

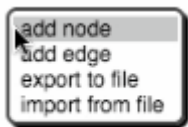
License: Copyright © Monash University, unless otherwise stated. All Rights Reserved.

Key points

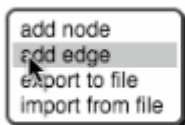
A graph or network consists of nodes and edges.



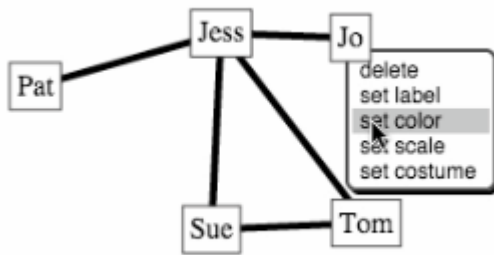
To create a node in Edgy, with your mouse over the stage, use the right-click menu and select 'add node'.



To create an edge in Edgy, with your mouse over the stage, use the right-click menu and select 'add edge'.



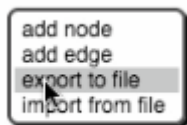
To set the colour of a node or edge, mouse over the node or edge, use the right-click menu and select 'set colour'.



To set the scale (size) of a node or edge, mouse over the node or edge, use the right-click menu and select 'set scale'.



To save your graph to a .dot file, mouse over the stage, use the right-click menu and select 'export to file'.

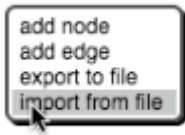


Here is an example of a graph saved as a .dot file. The 5 nodes are listed first, with the 5 edges underneath.

```
graph {
  "Jo" ;
  "Tom" ;
  "Jess" ;
  "Sue" ;
  "Pat" ;

  "Jo" -- "Jess";
  "Tom" -- "Jess";
  "Tom" -- "Sue";
  "Jess" -- "Sue";
  "Jess" -- "Pat";
}
```

To load a graph that is saved in a .dot file into Edgy, mouse over the stage, use the right-click menu and select 'import from file'.



Activity

Create your own social network on the stage in Edgy. Use nodes to represent yourself and some of your friends, and edges to represent friendships between people. Colour the node that represents you red. Add other colours or sizes to nodes and edges in your network to make it visually pleasing. Save your graph as a .dot file.

Summary

In this lesson, you have learnt how to use the Edgy interface to create your first graph. You have learnt how to:

- create a project in Edgy
- create nodes and edges in Edgy to build a network or graph
- set the colour and /or scale (size) of nodes and edges
- save your graph as a .dot file
- load a graph from a .dot file into Edgy

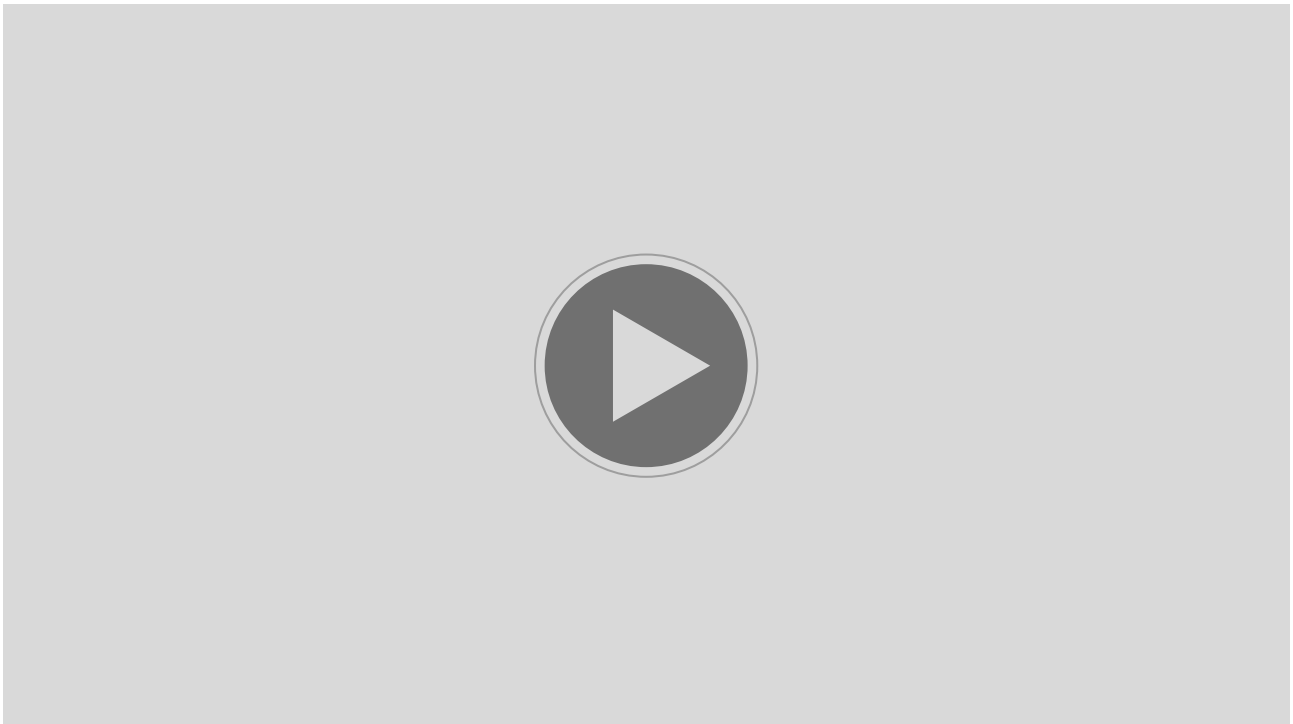
You can use the right-click menus to create nodes and edges on the stage to create a graph or network. You have customised the appearance of your graph by adding colour and images, setting the size of nodes and edges, and selecting an appropriate layout strategy.

1.2

A First Program: Using Blocks to Create a Social Network

In this lesson, you'll learn how to create a program to build a social network using Edgy. You'll be introduced to blocks, how to "snap" them together in sequence to make a program and how to run your program. Along the way, we'll learn about creating nodes and edges to form a network or graph.

Tutorial - Using Blocks to Create a Social Network



(https://www.alexandriarepository.org/wp-content/uploads/20161216110122/Edgy_a-first-program_v1-2_h_strm.mp4)

License: Copyright © Monash University, unless otherwise stated. All Rights Reserved.

Key Points

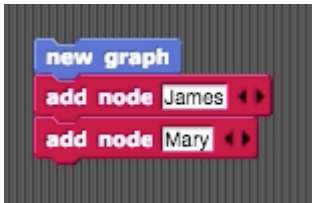
In Edgy, we can snap blocks together on the canvas to build programs that create graphs on the stage. Blocks are run (or executed) sequentially one after the other.



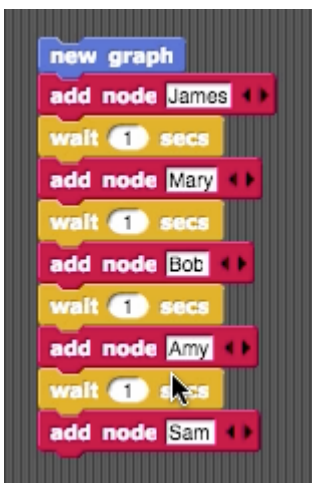
To add nodes to your graph, use an "add node" block.



To clear your graph on the stage, use an "new graph" block.



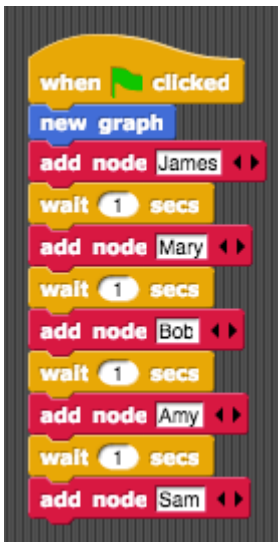
Use the "wait 1 secs" block to briefly pause (or slow down) the execution of your program.



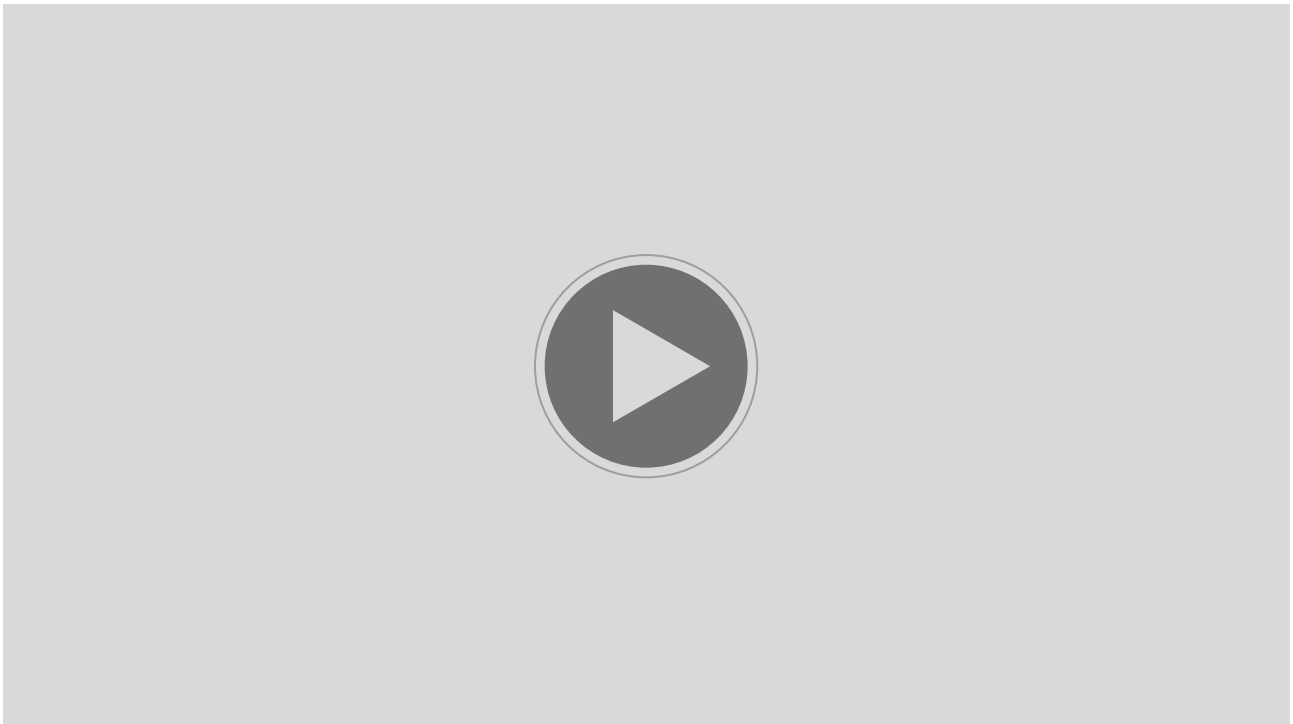
To add edges to your graph, use an "add edge" block with an "edge" block inside.



There are two ways to run your program in Edgy: either click anywhere on the blocks in your program, or click the green flag icon above the stage, if the program begins with a "when green flag clicked" block.



Tutorial - Modifying your Social Network

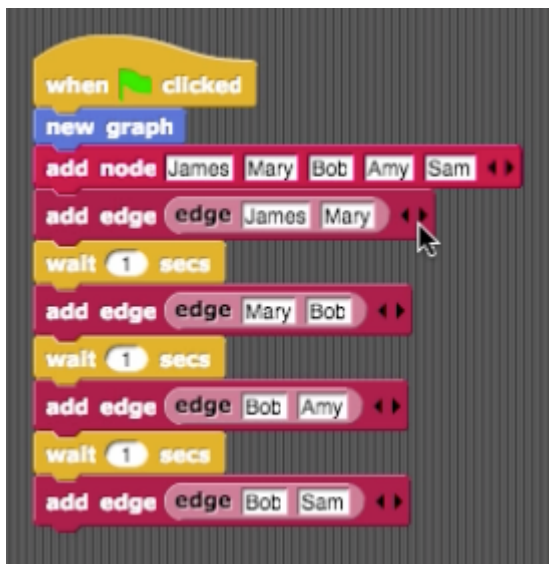


(<https://www.alexandriarepository.org/wp-content/uploads/20161218093846/first-program-part2.mp4>)

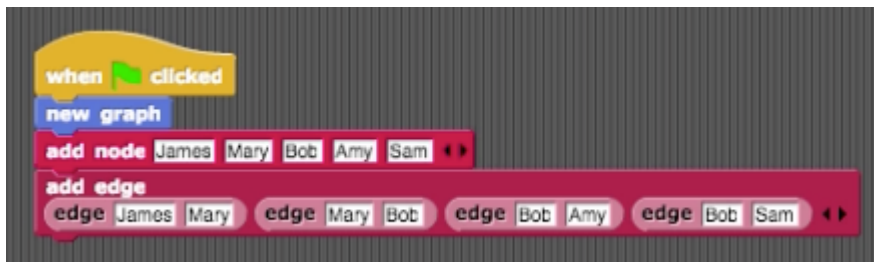
License: Copyright © Monash University, unless otherwise stated. All Rights Reserved.

Key points

We can add multiple nodes using one "add node" block by clicking on the arrows on the right side of the block to add more node inputs.



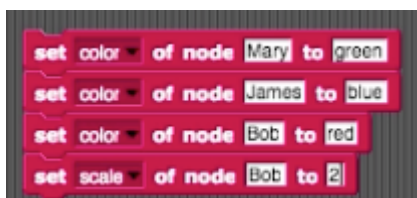
We can add multiple edges using one "add edge" block by clicking on the arrows on the right side of the block to add more edge inputs.



If you add an edge between nodes that do not exist, they will automatically be created when the edge is created.



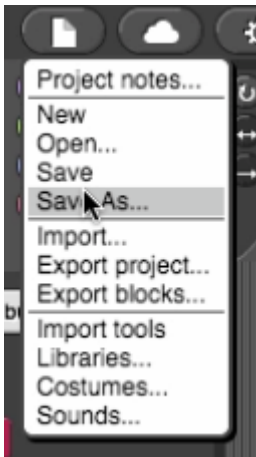
To set the colour or scale (size) of a node use the "set property of node to" block.



To set the colour or scale (size) of a edge, use the "set property of edge to" block.



To save your program as a project in the browser's cache, go to the "File" menu and select "Save As..." and give your project a name. You can close the web browser, and as long as you don't clear the web browser's cache, your project will still be there.



To save your program as an xml file, go to the "File" menu and select "Export project...". The xml file can be loaded into Edgy on other machines or shared with others.



Start Quiz (<https://www.alexandriarepository.org/app/WpProQuiz/94>)

Activity

Create a program using blocks in Edgy to build a graph that represents your local train or public transport network. This doesn't need to be complete or list all possible stops or stations, just give a few significant stops or stations that you might travel through, to or from. The nodes are stations or stops and edges are the routes that connect them. Colour your nearest station or stop blue.

Summary

Congratulations! Now you know how to create a program using Edgy. You have seen how to find blocks to create your program, drag them onto the canvas and snap them together with other blocks to create a program.

In this lesson you have learnt how to:

- add nodes to your graph using an "add node" block
- add edges to your graph using an "add edge" block
- use the "new graph" block to clear the stage and start a new graph
- use the the "wait 1 secs" block to pause or slow down the execution of your program
- run your program by clicking on it or the green flag icon, if you've used the "When green flag clicked" block at the start of your program
- set the colour or scale (size) of a node or edge using the "set property of node/ edge to" block
- save your program (or project) either in your browser's cache or as an xml file on your computer's hard drive

In this lesson we introduced you to some useful blocks for building networks or graphs from nodes and connecting these nodes using edges. You have also seen how to make your graph visually interesting by setting the colour and/or scale of the nodes and edges. You can save your project to use it again later or share with others.

1.3

Generating a Social Network: Processing Lists and Graphs

In this lesson, we will look at how you can create a graph in Edgy concisely and easily, just using one add-node block and one add-edge block. To achieve this, we'll use lists containing the nodes and edges to create and a for-each loop. We will also introduce another kind of for loop, which runs a specified number of times, with a counter keeping track of the number of loops. You will find these useful for a wide range of applications, in which blocks are called repeatedly a certain number of times, or until a certain condition is met.

Tutorial - Generating a Social Network

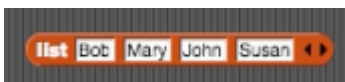


(https://www.alexandriarepository.org/wp-content/uploads/20161216110148/Edgy_lesson3_strm.mp4)

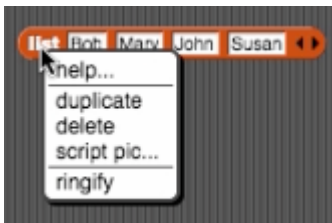
License: Copyright © Monash University, unless otherwise stated. All Rights Reserved.

Key Points

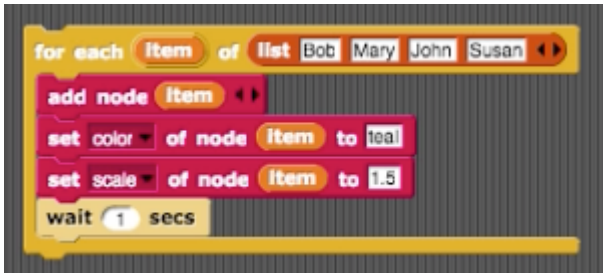
To create a list in Edgy, use the list block under the Variables tab.



To bring up a help window for a block, hover over the block and right click, then select help.



You can use a "for each item of ..." block to loop (or iterate) over the values in a list, and execute blocks of code, once for each item in the list. In this example we add a node, then set its colour and scale, for each item in our list.



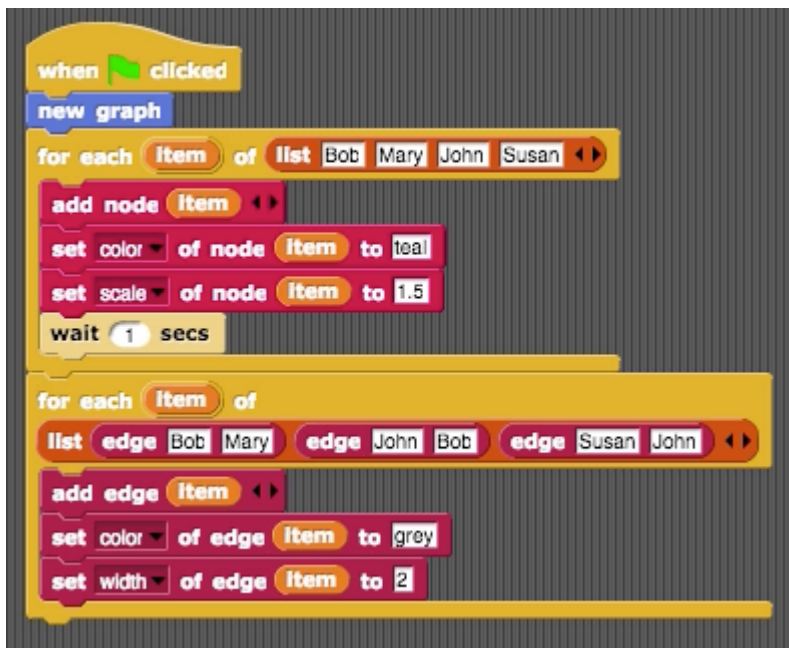
We can also create lists of edges using the list block in Edgy.



And we can use a "for each item of ..." block to loop over a list of edges. In the example below, we add an edge, then set its colour and width, for each edge in our list.



If we put our for loops together, we can create a program to generate a social network from lists of nodes and edges.



Activity

Think of your own social network and write a program to generate a graph that represents this using lists and loops. Colour the node that corresponds to yourself in the social network.

Tutorial - Generating a Path Graph

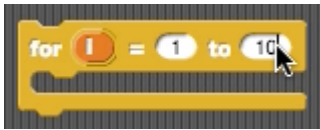


(https://www.alexandriarepository.org/wp-content/uploads/20161216110202/Edgy_lesson3-part2_strm.mp4)

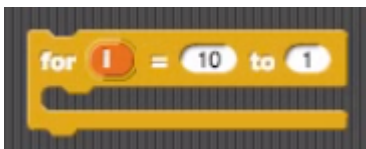
License: Copyright © Monash University, unless otherwise stated. All Rights Reserved.

Key Points

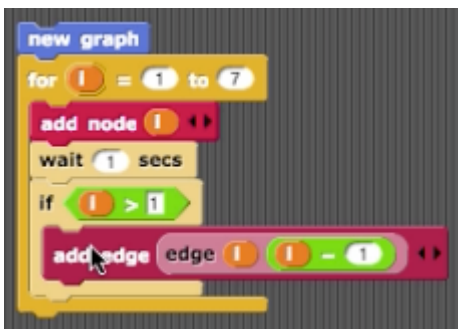
We can use a "for i in 1 to 10" loop block to repeatedly execute blocks of code a specific number of times. The value of i is incremented each time the loop is run.



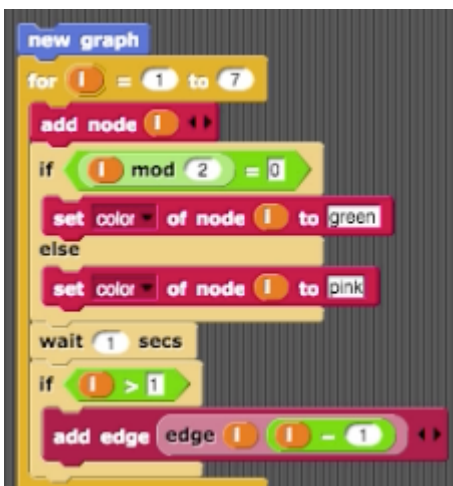
You can also loop from a larger value to a smaller value of i. In this case, the value of i is decremented each time the loop is run.



We can use a "for i in 1 to 10" loop block to create a path graph of the nodes named 1 to 7 and edges connecting them in order. To test whether a block should be executed, use an "if ..." block. In the example below, we use an "if ..." block to only add an edge if a certain condition and ensure that we create one less edges than nodes (i.e. our path graph has 7 nodes, and 6 edges connecting them).



To ensure that a block runs only in certain conditions, we can use an "if ..." block. If we want to run an alternative block, in the case that the condition is not met, we can use an "if ... else ..." block. In the example below, we use an "if ... else ..." block to colour even numbered nodes green, and odd numbered nodes pink.



Start Quiz (<https://www.alexandriarepository.org/app/WpProQuiz/126>)

Summary

In this lesson, you have been introduced to some new blocks in Edgy that you can use to control the flow of your program. In previous lessons, we have executed blocks one after another (sequentially), but now we have learnt how to repeatedly execute blocks using loops and how to control whether a block is executed using conditional blocks. Along the way you have also used lists and simple mathematical operators.

In this lesson you have learnt how to:

- create lists using the "list" block
- get help for a block by hovering over it, right-clicking and selecting "help"
- use conditional "if ..." blocks which control whether the blocks inside are executed
- use another conditional "if ... else ..." block to specify an alternative block to run if the if condition is not met
- use some simple mathematical operators, such as minus, modulo, greater than and equals.

You have also seen that using loops (or iteration) results in fewer repeated blocks in your program. You have learnt how to snap together the block that should be repeated inside a C-shaped loop block. The two different loop blocks we have used are:

- Loops over a list of values using foreach blocks. This block iterates over a list of values, performing the same action or actions for each item in the list.
- Loops with counters using for i in ... blocks. This block uses i as a counter to keep track of the number of times the blocks inside the loop should run. Every time the loop is run, the value of i changes by one (either incrementing or decrementing).

In this lesson we have created a social network from lists of nodes and edges, using for loops. We have also used for loops with counters, and used these to create a numbered path graph with nodes of alternating colours.

1.4

Exploring your Social Network

In this lesson we're going to explore our social network and create a program in Edgy to find out more about the people in it. You will see how to build a simple program to count the number of boys and girls in a social network. You will be introduced to variables, and learn how to use them in your programs to store and update intermediate values for later use. You will then be able to use variables to answer questions like; "Who is the person with the shortest name?", "How many people have names starting with the letter A?" or "Who has the most friends?".

Tutorial - Exploring your Social Network



(https://www.alexandriarepository.org/wp-content/uploads/20161216110218/Edgy_lesson4_v2_strm.mp4)

License: Copyright © Monash University, unless otherwise stated. All Rights Reserved.

Please download the attached file below (boys and girls network), containing an example social network, to follow along with the video tutorial. You can then right click in the stage and select "import from file" to import the social network into your copy of Edgy.

[boys and girls network](https://www.alexandriarepository.org/wp-content/uploads/boys-and-girls-network.txt) (<https://www.alexandriarepository.org/wp-content/uploads/boys-and-girls-network.txt>)

Key Points from the Video

Variables

A variable is a place where you can store a value to use again later. You can think of it like a labeled container. A variable can be a number, text, or any other value in Edgy.

In for loops, "i" and "item" are also variables. They store the current loop counter or list values

respectively. These variables are set to take on a different value each time the loop executes in a program.

- To create a variable in Edgy, click on "Make a variable" under the variables tab. It is important to give your variables meaningful names. This avoids confusion for anyone trying to read and understand your program.



- To set or update the value of a variable, use a set block.



- You can show or hide a variable on the stage, using the check box next to the variable name under the variables tab. This is useful if you want to see the value of a variable change as your program executes.



- You can also use the "show variable" and "hide variable" blocks to control the display of a variable on the stage.



- Click on "Delete a variable" and select the variable name, to delete a variable when you no longer need it.



- Use the "change variable by ..." block to update the value of a variable.



Nodes

The "all the nodes" block can be used to get a list of all the nodes in the current graph. Clicking on this block will evaluate it, and pop up a dialog with the list of all the nodes.

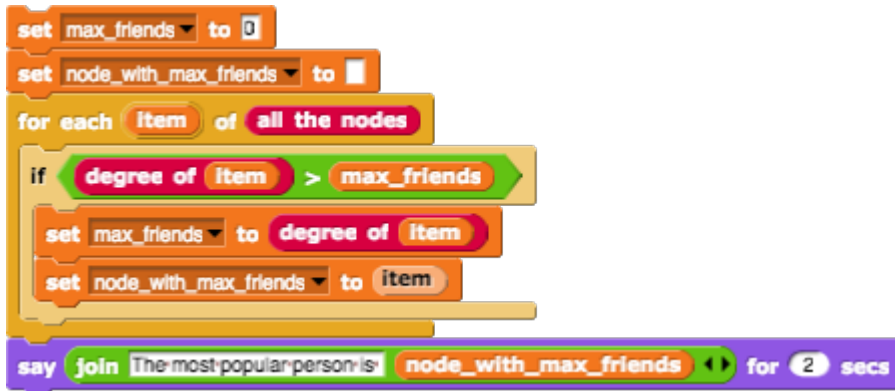


Operators

You can use operators to update, manipulate and compare variables, just as you would raw values in Edgy. E.g. `num1 - num2` or `join word1 word2`.

Examples

You can use variables to write programs to answer lots of questions about your social network. Here's a program that uses two variables to find the most popular person in a social network. Note that the "degree of ..." block, reports the number of edges that connect to a given node.



Activities

1. Build a program in Edgy to find the person in your social network with the longest name. You can use the example social network attached above in this learning module. Hint: you can use the "length of ..." block, to work out the length of a name - `length of world`.
2. How many people have names that start with the letter J? Use the example social network given above, and create a program in Edgy using variables to answer this question. Hint: The following block might be useful, to get the first letter of a word - `letter 1 of world`.

Start Quiz (<https://www.alexandriarepository.org/app/WpProQuiz/130>)

Summary

In this lesson you have seen how to create and update variables to store counts of the numbers of boys and girls in a social network. Variables can be used to store intermediate values in your program for later use. They can store numbers, text or any other value in Edgy and they can be manipulated and compared just as raw values would be.

You have learnt how to:

- create a variable
- set (or assign) its value
- show and hide a variable on the stage
- update the value of a variable
- delete a variable

You have also seen and created programs using variables, for loops, conditionals, and operators, to answer a variety of questions about the people in your social network. Now you can write simple

programs that answer questions like; "Who is the person with the shortest name?", "How many people have names starting with the letter A?" or "Who has the most friends?".

1.5

Who is friends with who?

In this lesson, we will look at the friendships in our social network. As a social network grows in size, the connections between friends are harder to see just by looking at the graph. We will write some simple programs in Edgy to discover which friends are shared by two people, and to visualise the spread of news between friends. Along the way, we will learn about lists in Edgy: how to create lists, add items to them and ask questions about the items they contain.

Tutorial - Who is friends with who?



(https://www.alexandriarepository.org/wp-content/uploads/20161216110237/Edgy_lesson5_v2_strm.mp4)

License: Copyright © Monash University, unless otherwise stated. All Rights Reserved.

Please download the attached file below (friends), containing an example social network to follow along with the video tutorial. You can then right click in the stage and select "import from file" to import the social network into your copy of Edgy.

[friends](https://www.alexandriarepository.org/wp-content/uploads/friends.txt) (<https://www.alexandriarepository.org/wp-content/uploads/friends.txt>)

Key Points from the Video


Lists


A list is an ordered collection (or sequence) of items. Lists can contain duplicates. The items in a list can be text, numbers or any other value in Edgy.

- To create a list in Edgy, use the list block under the variables tab. Use the black arrows to add or

delete items: 

- To create an empty list in Edgy, take a list block and remove all of the input boxes: 


- To get the length of a list use a "length of ..." block: 

- To add an item to a list (at the end of the list), use the "add thing to ..." block: 

- Use the "contains" block to check whether an item is present in a list:



- To get the first item of a list, use "item 1 of ...". You can also use this block to get the last item of a

list, or any other item by its position in the list. For example: 

or 

- There are lots of other useful blocks for creating, manipulating and querying lists in Edgy. You can find them under the variables tab.



- To create a list as a variable in Edgy, after you have created the variable, simply set the value of the variable to be a list using a "set variable_name to ...". For example:



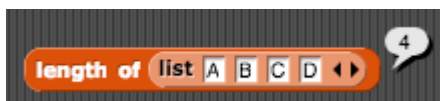
Nodes

The "neighbours of ..." block under the Nodes tab reports a list of the nodes that have an edge connecting them directly to the given node.



Reporter blocks

Oval-shaped blocks in Edgy are "reporter" blocks. These blocks are called "reporters" because when they're run, instead of carrying out an action, they report a value that can be used as an input to another block. If you drag a reporter block on to the canvas by itself and click on it, the value that it reports will pop up next to it.



Boolean Values

The values **true** and **false** are called boolean values. Boolean values are the result of conditional statements (like those used in an "if" block), which allow different actions and change control flow depending on whether a boolean *condition* evaluates to true or false. For example:



Boolean values can be combined into more complicated expressions using the "and", "or" and "not" operators. For example:



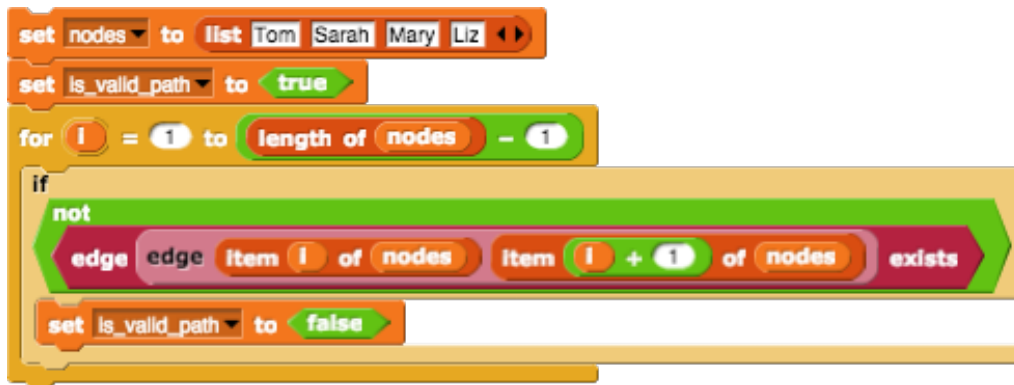
Help Tip

If you can't remember which tab (coloured palette) a block is under, but you remember at least part of its name, type Control-F and enter the name in the text input box that appears in the palette area.



Examples

Lists are very useful for collecting together a number of items in a program. Here's a program that checks whether a list of nodes, represents a valid path in the current graph (i.e. whether there is an edge that connects each of the adjacent items in the list). The boolean variable "is_valid_path" is set to false when no edge between one node and the next exists (the last node in the path is not checked as no "next" node exists).



Start Quiz (<https://www.alexandriarepository.org/app/WpProQuiz/135>)

Activities

1. Imagine a social network in which the girls are coloured pink and the boys are coloured blue. Write a program in Edgy to build up two lists - one containing the boys and one containing the girls.
2. Write a program in Edgy to create a list of all the people in a social network who's names begin with the letter "J". Hint: The following block might be useful, to get the first letter of a word -

letter 1 of word

Summary

In this lesson you have seen how to use lists to discover and visualise the friendships in a social network. A list is an ordered collection of items that may contain duplicates. Lists can also be stored as variables for accessing later in your programs.

You have learnt about lists in Edgy - how to:

- create a list, either empty or containing items
- get the length of a list
- add items to a list
- check whether an item is a member of a list
- access an item by its position in the list
- store a list as a variable

You have also learnt about:

- oval-shaped "reporter" blocks. These blocks report a value that can be used as an input to another block.
- the boolean values "true" and "false", and combining these using the "and", "or" and "not" operators.

You have used variables, lists, for loops, booleans and various operators to build up programs in Edgy. You have seen how to create and add items to lists and how to access items in a list. Now you can write programs to create lists of nodes, and to use lists to explore the nodes in a network.

1.6

Making Friend Recommendations

In this lesson, we will write a program to make friend recommendations to a person in our social network. We'll look at a simple algorithm for finding potential new friends, and use nested loops to generate a list of friend recommendations. A nested loop is a loop within another loop: an inner loop within an outer one. We will also see more complicated boolean expressions using the operators "and", "or" and "not".

Tutorial - Making Friend Recommendations



(https://www.alexandriarepository.org/wp-content/uploads/20161216110258/Edgy_lesson6_strm.mp4)

License: Copyright © Monash University, unless otherwise stated. All Rights Reserved.

Please download the attached file below (friends), containing an example social network, to follow along with the video tutorial. You can then right click in the stage and select "import from file" to import the social network into your copy of Edgy.

[friends](https://www.alexandriarepository.org/wp-content/uploads/friends1.txt) (<https://www.alexandriarepository.org/wp-content/uploads/friends1.txt>)

Key points from the Video

Nested Loops

A nested loop is a loop inside another loop: an inner loop inside an outer loop. For each iteration of the outer loop, the inner loop will run to completion. For example:



In our example, we have an outer loop over 3 items (the 3 friends of James), with inner loop that runs, 2, 7 and 6 times respectively for each of the outer loop items. This means that the code inside the inner loop will run 15 times in total (2 + 7 + 6).

Note that nesting loops may result in duplicated variable names in your program (e.g. the variable called "item" in a "for each item in ..." block, or the variable called "i" in the "for i in 1 to 10" block). If you have two variables with the same name in your program, you should rename (at least) one of them to avoid confusion to those trying to read, understand or modify your program.

Boolean expressions

We can use boolean operators "and", "or" and "not" to combine expressions that evaluate to either "true" or "false".

- For example, the following expressions can report either "true" or "false":



- A "not" block can be used to reverse the truth value of an expression. For example:



- The "and" operator can be used to combine two boolean expressions (expressions that evaluate to "true" or "false"), such that it evaluates to "true" if both sub-expressions are "true", and "false" otherwise. For example:



- The "or" operator can also be used to combine two boolean expressions. It evaluates to "true", if either sub-expression is "true" or both are "true", and "false" otherwise. For example:



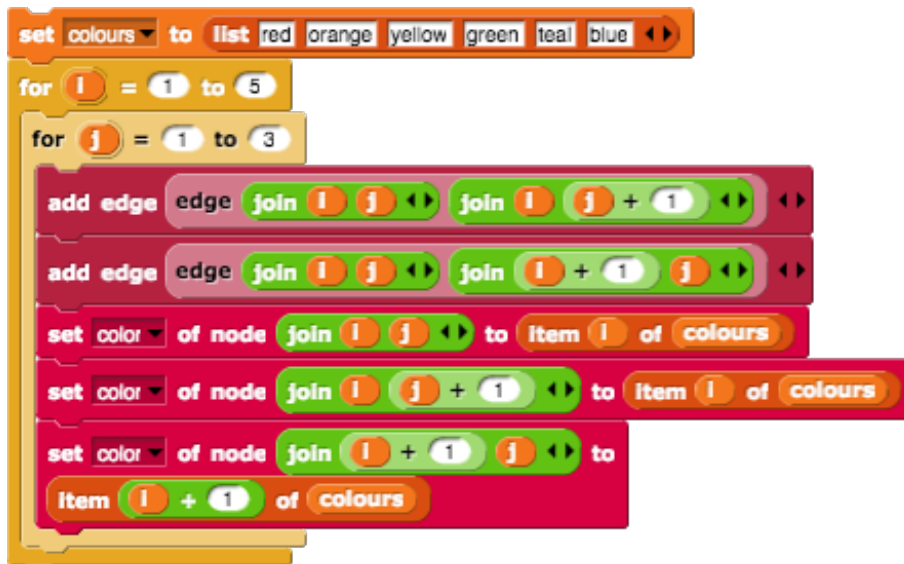
Variables

We use variables to store values that are used or calculated in more than one place in our program. For example, the string "James" is stored as a variable called "person_to_recommend_to" in the video above. This avoids duplication of code and ensures that any modifications to this value only need to occur in a single place in our program.

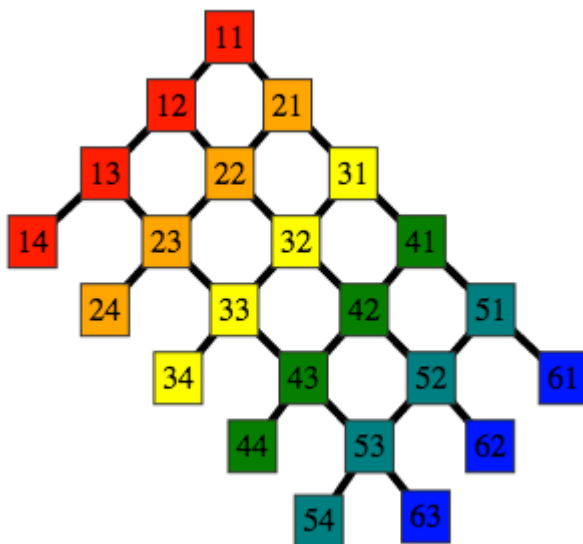
Start Quiz (<https://www.alexandriarepository.org/app/WpProQuiz/136>)

Example

Here's a program in Edgy to generate a coloured grid graph. It uses nested loops - the outer loop produces a row number and the inner loop generates a column number. Each node is named by joining the row number followed by the column number. A list of colours is used to create a rainbow effect across the rows.



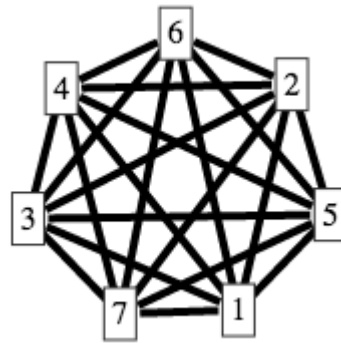
And this is the graph that it produces:



(<https://www.alexandriarepository.org/wp-content/uploads/grid-colouring.png>)

Activity

1. Write a program using nested loops in Edgy to generate a complete graph with 7 nodes. A complete graph is a graph in which every distinct pair of nodes is connected by an edge, like the



one pictured below.

- Given the graph shown on the left below, write a program in Edgy using nested loops to create edges to connect every pair of different coloured nodes (this is called a bipartite graph).



becomes:-

Summary

In this lesson you have learnt about nested loops in Edgy and seen examples of how to use them to generate friend recommendations (lists of friends of friends) . You have also seen an example of using nested loops to generate and colour a grid graph.

- A nested loop is an inner loop nested inside an outer loop.
- The inner loop will run to completion for each iteration of the outer loop.
- When nesting loops of the same kind, the loop variables in the inner and outer loops may have the same name. In this case, you should rename at least one of these variables, to avoid confusion.

You have also seen further examples of boolean expressions, which evaluate to "true" or "false". Boolean expressions are used to determine whether the blocks inside a conditional ("if") block should be run. You have learn about the following boolean operators:

- "and" - combines two boolean expressions. "True" if both sub-expressions are "true", "false" otherwise.
- "or" - combines two sub-expressions. "True" if either one or both of the sub-expressions is "true", "false" otherwise.
- "not" - reverses the truth value of an expression (i.e. "true" becomes "false" and "false" becomes "true")

You have seen how to create more complex control flows in your programs by nesting loops with loops and using complex conditional expressions involving booleans. Now you are able to use these to generate friend recommendations in a social network and to generate graphs that make patterns like grid graphs or complete graphs.

1.7

Visualisations for Social Networks

In this lesson, we're going to write programs to produce various different visualisations of a social network. Our visualisations will add colour and scale to the nodes in our network, enabling us to highlight and represent certain aspects of the social network.

Tutorial - Social Network Visualisations

Part 1



(https://www.alexandriarepository.org/wp-content/uploads/20161216110321/Edgy_lesson7-part1_strm.mp4)

License: Copyright © Monash University, unless otherwise stated. All Rights Reserved.

Part 1 of the video tutorial presented a problem for you to solve in Edgy. Create a program to visualise the nodes in your social network, such that they are coloured and sized according to their degree (the number of friends they have). So, for example, all nodes with 5 friends are coloured red and scaled 2.5, the nodes with 4 friends are coloured orange and scaled 2, the nodes with 3 friends are coloured yellow and scaled 1.5, and so on. Please have a go at writing a program to create this visualisation in Edgy. Once you've finished, watch Part 2 of the video tutorial for a discussion of some possible solutions.

Part 2



(https://www.alexandriarepository.org/wp-content/uploads/20161216110307/Edgy_lesson7-2_strm.mp4)

License: Copyright © Monash University, unless otherwise stated. All Rights Reserved.

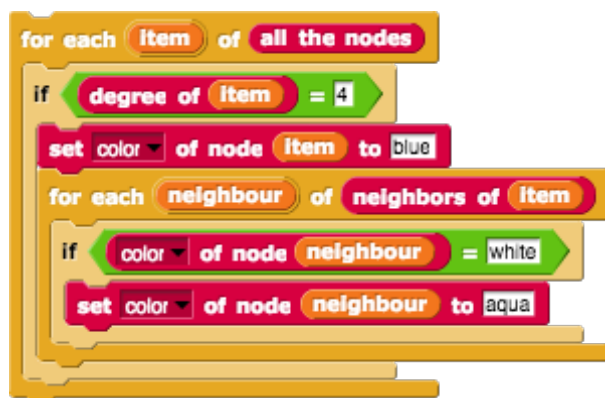
Please download the attached file below (social_network), containing an example social network, to follow along with the video tutorials. You can then right click in the stage and select "import from file" to import the social network into your copy of Edgy.

[social_network](https://www.alexandriarepository.org/wp-content/uploads/social_network.txt) (https://www.alexandriarepository.org/wp-content/uploads/social_network.txt)

Key points from the Videos

Nested loops

We use nested loops in our visualisation of sub-graphs containing a node that has exactly 4 friends. An outer loop is used to iterate through all of the nodes in the graph, and an inner loop loops over the friends of some of these nodes. For each node in the outer loop that had exactly 4 friends, we executed the inner loop 4 times, once for each of those 4 friends.



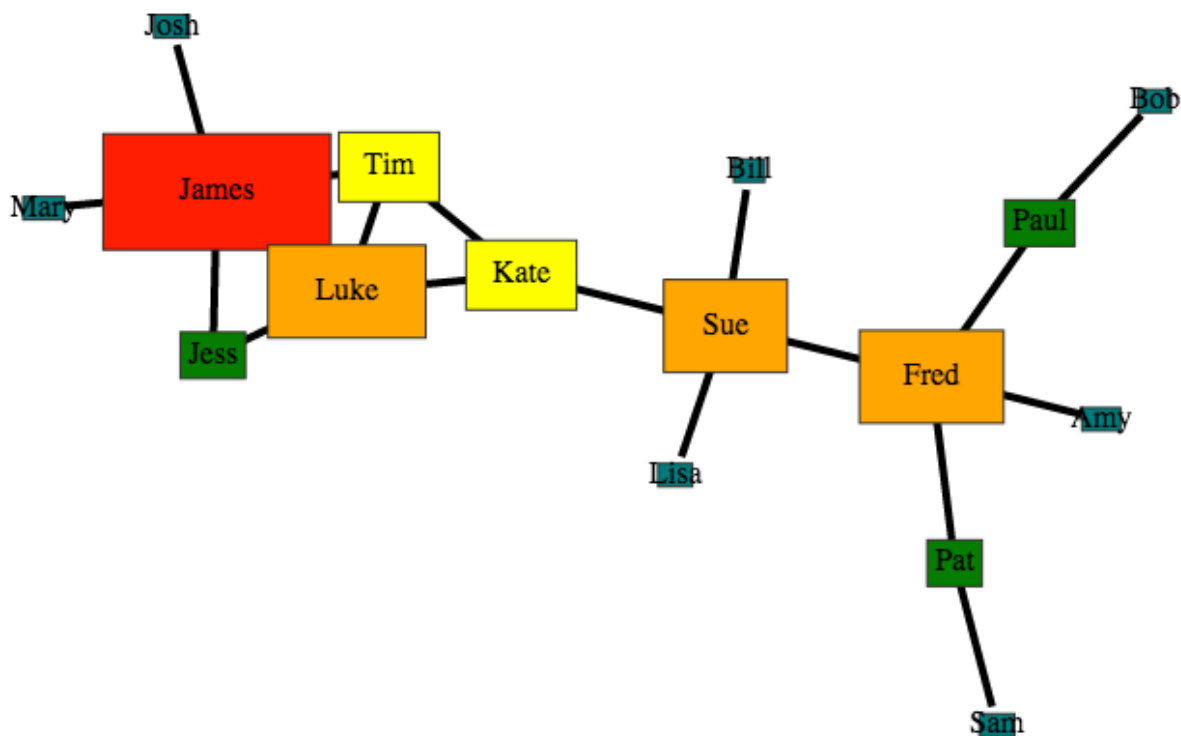
Debugging

Debugging is the process of locating and fixing errors (or bugs) in your program. If your program is not doing what you expect, there are various approaches that you might try in order to diagnose the problem so that you are able to fix it.

- Use "wait 1 secs" blocks to slow down the executing of your program, so that you can see what changes it is making to the objects on the stage or to the values of variables.
- Use the pause button to pause your program as it runs and view the values of variables (or the "pause all" button) or the state of the objects on the stage.
- Separate a block or blocks from the rest of your program and run them on their own, in order to confirm that they are doing what you expect.

Visualising nodes by degree

We discussed an exercise, in which we created a visualisation for our social network that coloured and scaled nodes according to their degree (the number of friends they have). For example, a node with 5 friends is coloured red and scaled to 2.5, a node has 4 friends is coloured orange and scaled to 2, and a node with 3 friends is coloured yellow and scaled to 1.5. Here's an example of a social network visualised this way:



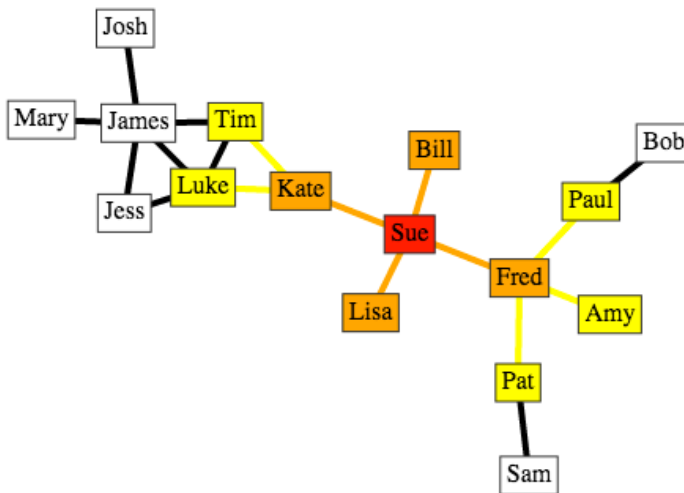
We looked at 2 programs to create this visualisation. In both programs the scale of a node was calculated by dividing the degree of a node by 2.

1. The first program used a list of colours and the degree of the node was used to get the item at this index (or position) in the list of colours. This solution is very concise.
2. The second program used a series of nested "if... else..." blocks to check the degree of a node and assign it a colour accordingly. This solution is longer and required some repeated code blocks.

There are often a number of possible programs that will solve the same problem. Have you come up with other solutions?

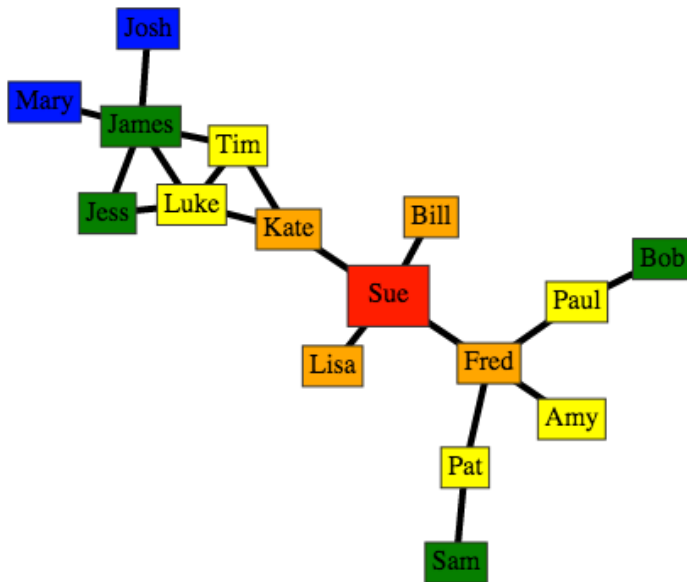
Activities

1. Colour all the nodes whose names start with the letter "J" red, and all the nodes whose names start with the letter "S" blue.
2. Write a program to visualise one node "X" as the centre of the network and colour it red, and then colour all X's direct friends in orange and, colour all of the friends of X's friends (who are not already X or a direct friend) yellow. You might like to colour the edges as well. Here's an example, where "Sue" is the centre of the network:



Challenge

Generalise the second activity above, to colour the whole network using a list of colours - working outwards from a centre node, level by level. Colour all the nodes in the network according to the shortest number of edges between them and the central node. In the example below, "Sue" is the central node coloured red. Her direct friends are coloured orange. People she is friends of friends with are coloured yellow. And so on, until every node in the network is assigned a colour according to the number of "degrees of separation" between them and Sue.



Summary

You can write programs to do lots of fun and useful visualisations of your network. They are easy ways to make a network more visually interesting and highlight certain aspects of the data that is presented in you network.

1.8

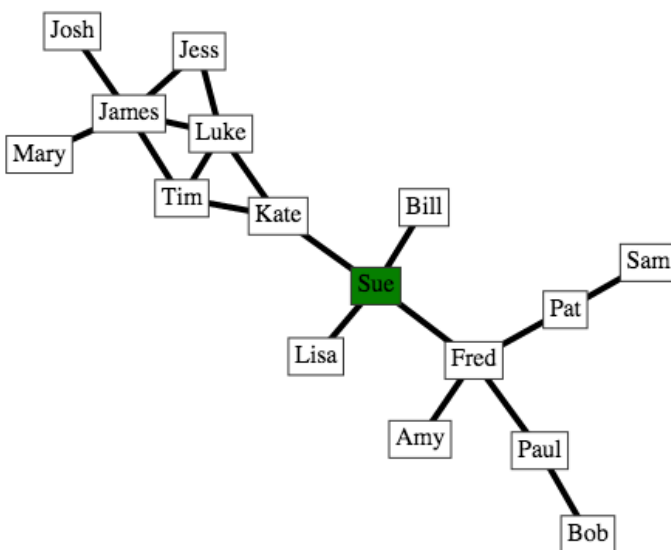
Visualising the Spread of News in a Social Network

In this lesson, we're going to look at a visualisation of news spreading through a social network. We will use colour to indicate that a node has received the news and friends in the social network will be able to tell the news to one another. We will also investigate how the structure of a network affects the speed that news will spread. We will build a complex program, utilising many of the different programming constructs in Edgy that you have been introduced to over the course of the "Programming with Edgy" syllabus. We will also discuss strategies for building and debugging complex programs.

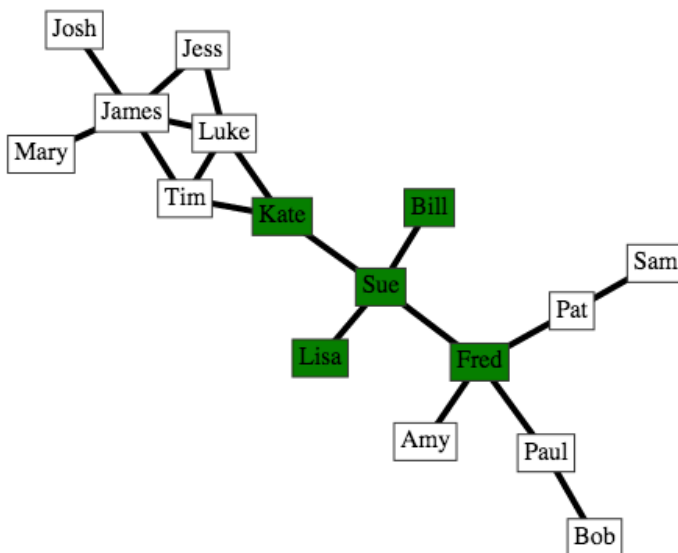
Tutorial Task

In the video tutorial below, we're going to look at a program to visualise the spread of news in a network. Before you watch the tutorial, however, please have a go at this task yourself.

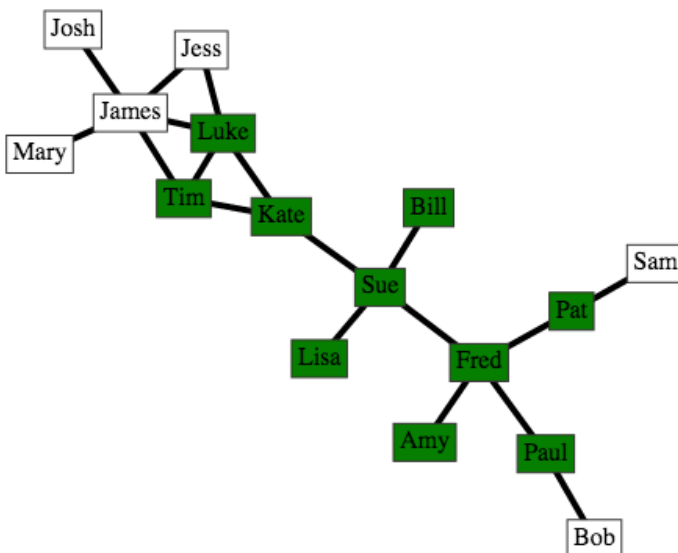
Your program should start with one node in the social network, which is coloured, to indicate that this person has heard some news.



This person then tells the news to their friends...



..., who tells their friends...



...and so on, until all the people in the social network have heard the news. Your program should colour the nodes in the network to indicate when a person has heard the news.

Please download the attached file below (social_network), containing an example social network to use and to follow along with the video tutorials. You can then right click in the stage and select "import from file" to import the social network into your copy of Edgy.

[social_network](https://www.alexandriarepository.org/wp-content/uploads/social_network.txt) (https://www.alexandriarepository.org/wp-content/uploads/social_network.txt)

Tutorial - Visualising the Spread of News in a Social Network

Part 1 - Creating the Program



(https://www.alexandriarepository.org/wp-content/uploads/20161216110339/Edgy_lesson8-1_strm.mp4)
License: Copyright © Monash University, unless otherwise stated. All Rights Reserved.

Part 2 - Running the Program on Different Networks



(https://www.alexandriarepository.org/wp-content/uploads/20161216110357/Edgy_lesson8-2_strm.mp4)
License: Copyright © Monash University, unless otherwise stated. All Rights Reserved.

Example graph structures:

Please download the files below, containing example graph structures to run your visualisation on. You can then right click in the stage and select "import from file" to import them into your copy of Edgy.

[path_graph](https://www.alexandriarepository.org/wp-content/uploads/path-graph.txt) (https://www.alexandriarepository.org/wp-content/uploads/path-graph.txt)

[disconnected_graph](https://www.alexandriarepository.org/wp-content/uploads/disconnected-graph1.txt) (https://www.alexandriarepository.org/wp-content/uploads/disconnected-graph1.txt)

[circle_graph](https://www.alexandriarepository.org/wp-content/uploads/circle-graph1.txt) (https://www.alexandriarepository.org/wp-content/uploads/circle-graph1.txt)

[circle_graph with cross links](https://www.alexandriarepository.org/wp-content/uploads/circle-graph-with-cross-links1.txt) (https://www.alexandriarepository.org/wp-content/uploads/circle-graph-with-cross-links1.txt)

[complete_graph](https://www.alexandriarepository.org/wp-content/uploads/complete-graph.txt) (https://www.alexandriarepository.org/wp-content/uploads/complete-graph.txt)

[grid_graph](https://www.alexandriarepository.org/wp-content/uploads/grid-graph.txt) (https://www.alexandriarepository.org/wp-content/uploads/grid-graph.txt)

[graph with friend clusters](https://www.alexandriarepository.org/wp-content/uploads/graph-with-friend-clusters.txt) (https://www.alexandriarepository.org/wp-content/uploads/graph-with-friend-clusters.txt)

Key Points from the Video

Repeat Until loops

A "repeat until" loop repeatedly executes the code inside it until the specified stopping condition is met. The example below uses a "repeat until" loop, to generate a list of numbers starting at 1 and counting by four. The loop stops once the current number is greater than 16.



gives:

Building a Program

Building up a non-trivial program can be complex and a little daunting. It is sometimes helpful to employ the following strategies in order to debug your program and diagnose whether it is correct.

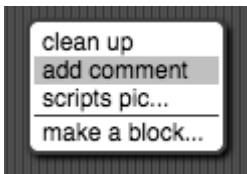
- Work through an example on paper and run this example through your program, ensuring the steps in your program follow your example.
- Select a few examples that you can use to test your program in a variety of situations (e.g. different graph structures, different input or variable values) to ensure that your program is robust to these differences.
- Extract a particular block or blocks from your program temporarily and try them in isolation, to ensure that a particular part of your program does what you think it should do.
- Use "say ... for ... secs" blocks and/or show your variables on the stage, so that you can verify any intermediate values that your program calculates or uses.
- Use "wait 1 secs" blocks to slow down the execution of your program, so that you can see the values of variables and any modifications that it has made to the network on the stage.
- Pause your program during execution in order to check intermediate values or modifications that

your program has made to the graph on the stage.

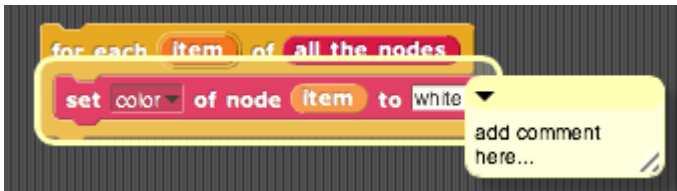
Comments

Comments are human readable annotations attached to all or parts of a program, which will be ignored by the computer when it runs the program. Comments can provide a brief description or explanation for a block or blocks of code, to make the program easier to understand.

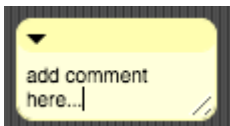
To add a comment to a program in Edgy, right click on the canvas and select "add comment".



Then drag and drop the comment panel to attach it to the block of code that it refers to.

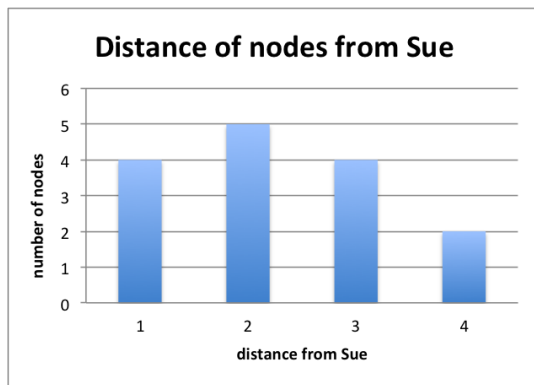


And click inside the comment panel to edit the text inside it.



Activities

1. Write a program to visualise the nodes in your social network by gender. Colour the nodes representing boys blue and the girls pink (or choose your own colours!). Hint: You will need to create a list (or lists) containing the boy and/or girl nodes.
2. Modify the program from the videos above, to calculate how many iterations it takes for the news to spread from "Sue" to "James" (for example). Hint: After each iteration to colour the next "level" of friends, you will need to check whether "James" has been coloured.
3. Perform some analysis of the program from the videos above, examining the number of people that the news reaches after each iteration to colour the next "level" of friends, starting with a given starting node. For example, in the example social network from the first video above:- if we start with Sue, after 1 iteration, the news will have spread to all 4 of Sue's direct friends, then after 2 iterations it will reach all 5 of the friends of Sue's friends who were not already coloured.



Select at least two different network structures from the example graphs given above and create a bar graph like the example above to represent this information for each one. Compare the two graphs and what they show about the speed that news spreads through the different networks.

Challenge

Modify the program from the videos above to find the best start node (or nodes). The best start node is the start node that will spread the news to reach all the nodes in the network in the shortest number of iterations - there may be more than one best start node in a network. Your program will need to try each possible start node in the network and count the number of iterations required to spread the news to all the nodes from each one. Then it will need to choose the start node (or nodes) that results in the smallest number of iterations to spread the news to the whole network.

Summary

In this lesson, you have seen how to build a complex program in Edgy to visualise the spread of news through a social network. We have used colour to indicate when a node has received the news, and analysed the effect that different network structures have on the speed that news spreads. Different networks and their characteristics have been used to model or represent the problem that we wish to analyse.

Throughout the "Programming in Edgy" syllabus, we have used social networks to introduce to Edgy and graph constructs that Edgy uses (nodes and edges). We represented people using nodes and friendships using edges. However, graphs such as those in Edgy, can also be used to represent many types of relations and processes in physical, biological, social and information systems. Many practical problems can be represented by graphs such as transport networks, the structure of websites, biological relationships, flow in physical networks (e.g electrical distribution, water flow etc.). Modelling these problems using graphs allows us to apply known solutions to graphing problems in these varied areas.

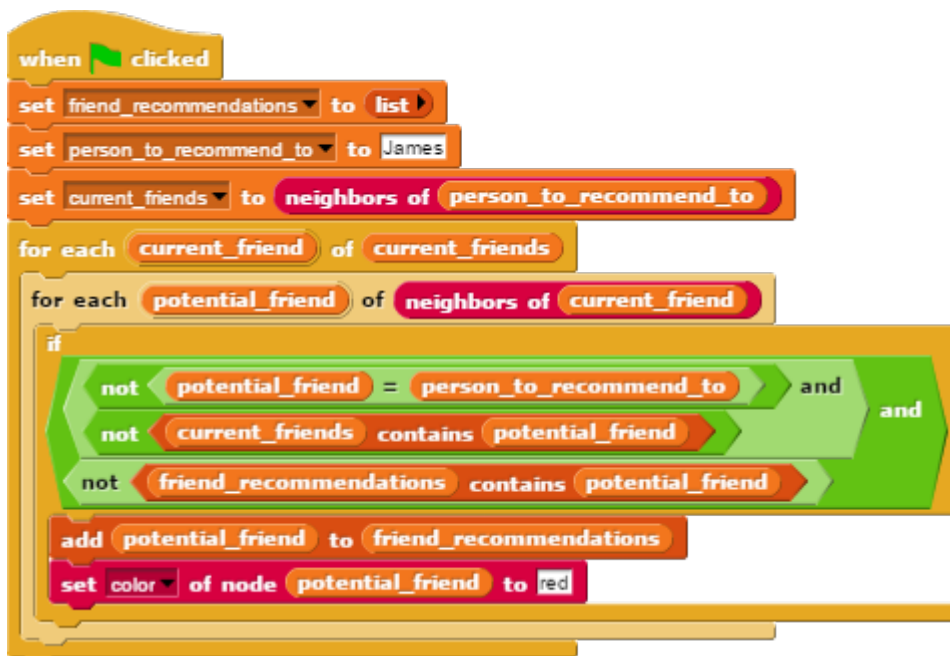
1.9

Appendix: From Edgy to Python

Once you're familiar with Edgy, you might like to do the same tasks using a conventional programming language, such as Python. If you're using Mac OS X or Linux, your system probably already has Python installed. Otherwise, you'll have to install it from the [Python website](https://www.python.org/downloads/) (<https://www.python.org/downloads/>).

Python

Python is a versatile programming language which lends itself to quick development of programs. We're using it here since it's easy to read and understand, and is also widely used, with a plethora of libraries available (such as those for dealing with graphs, collections, games development, web development, and many more). The goal of this tutorial is to show how to convert Edgy programs like this:



Into Python programs like this:

```
from edgy import *

# import the graph
G = nx.drawing.nx_pydot.read_dot('friends1.txt')

# the main program
friend_recommendations = []
person_to_recommend_to = 'James'
current_friends = G.neighbors(person_to_recommend_to)

for current_friend in current_friends:
    for potential_friend in G.neighbors(current_friend):
        if (potential_friend != person_to_recommend_to and
            potential_friend not in current_friends and
```

```

        potential_friend not in friend_recommendations):
    friend_recommendations.append(potential_friend)
    G.node[potential_friend]['color'] = 'red'

```

```

# show the results
print(friend_recommendations)
show(G)

```

While we're not quite ready to do such a conversion yet (but we will be soon!), you should already be able to see that there is a near one-to-one correspondence between Edgy blocks and Python code (and this is by no means a coincidence). For example, the block setting `friend_recommendations` to an empty list corresponds to line 7, and the block setting `person_to_recommend_to` to James corresponds to line 8.

First, we'll take a look at some basics.

Having installed Python, use the following commands to install the required libraries:

```

$ pip install networkx
$ pip install matplotlib
$ pip install pydotplus

```

Getting Started

Python provides a development environment called IDLE, which lets us run Python programs interactively. Open that up, and you should be greeted with a shell indicating your installed version of Python, ready for you to type Python code into.

Before doing anything at all, we need to import the libraries for graphs and plotting as follows:

```

>>> import networkx as nx
>>> import matplotlib.pyplot as plt

```

Now we can create a simple graph:

```

>>> G = nx.Graph()
>>> G.add_node("James")
>>> G.add_node("Mary")
>>> G.add_node("Bob")
>>> G.add_edge("James", "Mary")
>>> G.add_edge("Mary", "Bob")

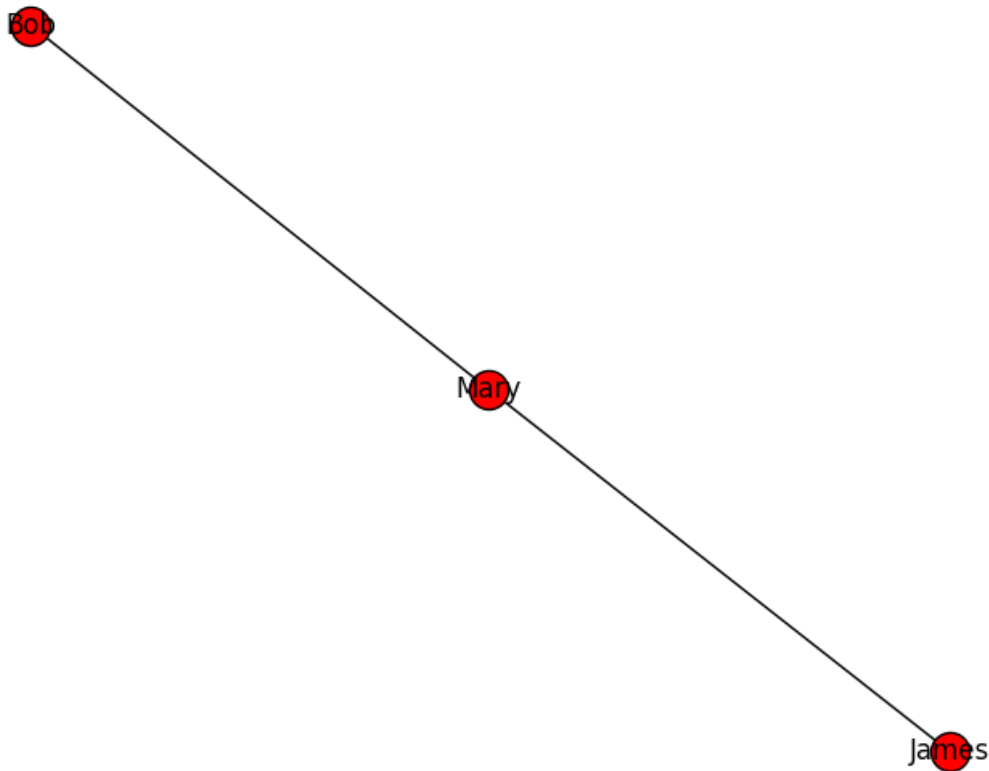
```

Now we can see the graph:

```

>>> nx.draw(G, with_labels = True)
>>> plt.show()

```



From Edgy to Python

So having played around a bit with manipulating a graph in Python, you begin to get a feel for how things work. Here are some of the fundamental conversions of Edgy blocks to Python code:

Control







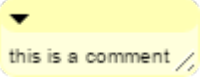


```

if a == b:
    foo(bar)
else:
    baz(bar)
E.g.
>>> if 10 == 5:
...     print('equal')
... else:
...     print('not equal')
...
not equal
  
```

Comparison is similar to Edgy, except that equality is tested using double equals signs ==

- is a equal to b? `a == b`
- is a not equal to b? `a != b`
- is a greater than b? `a > b`
- is a greater than or equal to b? `a >= b`
- is a less than b? `a < b`
- is a less than or equal to b? `a <= b`
- is an expression e false? `not e`

	<code>for item in collection:</code>
	<code>foo(bar)</code>
	<code>G = nx.Graph()</code>
	<code>G.add_node(1)</code>
	<code>G.add_edge(1, 2)</code>
	<code>print('Hello!')</code>
	<code># this is a comment</code>

E.g.
`>>> for item in ['A', 'B', 'C']:`
`... print(item)`
`...`
A
B
C

This defines the graph, and must be done before nodes/edges can be added.

You can set node attributes while adding a node:
`G.add_node(1, color='red')`

You can set edge attributes while adding an edge:
`G.add_edge(1, 2, color='red')`

Comments start with a # symbol and extend to the end of the line:

The end of this article contains a reference for translating many common Edgy blocks into Python.

A Helper Module

Edgy supports many built-in attributes, such as node x and y position, node/edge colour, edge width and edge labels. However, the libraries we're using in Python don't consider such attributes. In order to ease the transition from Edgy to Python, we have created a simple module which allows us to use some of the more important built-in attributes:

```
import networkx as nx
import matplotlib.pyplot as plot
from collections import deque
try:
    from Queue import PriorityQueue
except ImportError:
    from queue import PriorityQueue

def show(G, node_attribute = "id", edge_attribute = "label"):
    layout = nx.spring_layout(G)

    for v, data in G.nodes(data = True):
        if "x" in data:
            layout[v] = [data["x"], layout[v][1]]
        if "y" in data:
            layout[v] = [layout[v][0], data["y"]]

    node_colors = [G.node[v].get("color", "white") for v in G.nodes()]
    edge_colors = [G.edge[e[0]][e[1]].get("color", "black") for e in G.edges()]
    node_labels = dict((v, v if node_attribute == "id" else
G.node[v].get(node_attribute, v))
        for v in G.nodes())
```

```

edge_labels = dict((e, G.edge[e[0]][e[1]].get(edge_attribute, "")) for e in
G.edges())
nx.draw(G, layout, node_color = node_colors, edge_color = edge_colors)
nx.draw_networkx_labels(G, layout, node_labels)
nx.draw_networkx_edge_labels(G, layout, edge_labels)
plot.show()

```

The module provides us with a `show()` function that plots a graph with node positions, labels and colours like Edgy does. Note that the axes that our graph will be drawn on are such that greater y-values are higher up (as opposed to in Edgy where greater y-values are lower). By saving this as `edgy.py` we can make use of it in IDLE by opening the file, choosing `Run > Run Module`, then importing the the module:

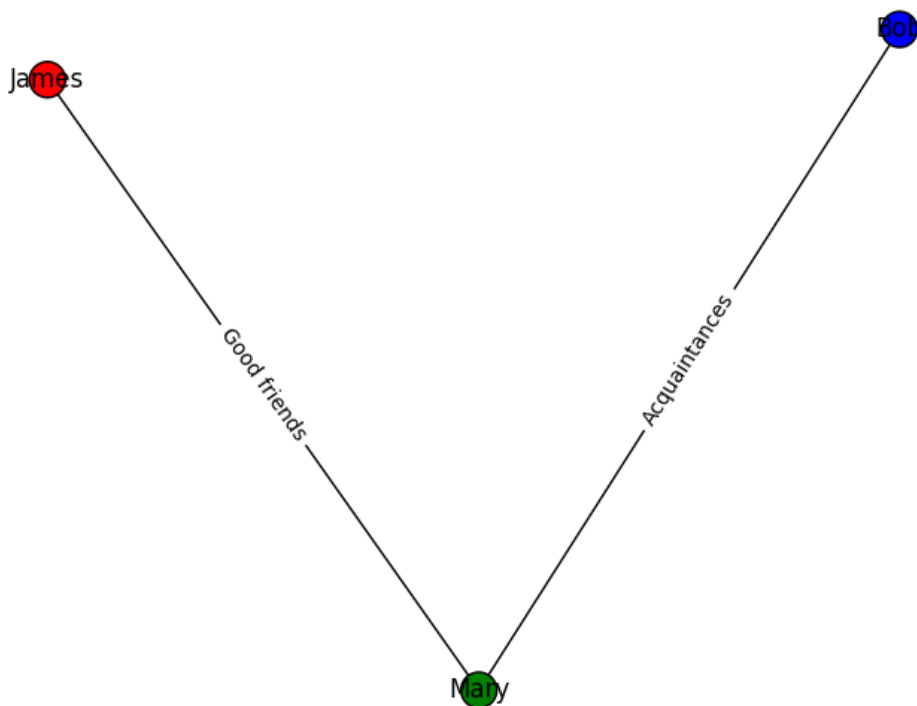
```

>>> from edgy import *
>>> G = nx.Graph()
>>> G.add_node("James", color='red')
>>> G.add_node("Mary", color='green')
>>> G.add_node("Bob", color='blue')
>>> G.add_edge("James", "Mary", label='Good friends')
>>> G.add_edge("Mary", "Bob", label='Acquaintances')

```

And now we can see our graph by simply calling:

```
>>> show(G)
```



We can also display custom node/edge attributes with:

```
>>> show(G, 'my_node_attribute', 'my_edge_attribute')
```

Keep in mind that this module is provided to ease the transition from Edgy to Python. It's not yet necessary to understand exactly what it does and why. For learning purposes, the module can be treated as a 'black box', which lets us draw our graph in a way that is convenient and familiar to us.

Once you have become more familiar with Python and the graph library NetworkX, you may wish to have a go at figuring out what goes on inside this module.

A Note on Syntax

While Edgy provides a way to quickly turn algorithmic thinking into working programs through its drag and drop system, conventional languages have another aspect to be concerned with: syntax. Syntax describes the rules and principles that govern a particular language. This is akin to having to use correct grammar in English.



Since programming language parsers are generally less flexible in terms of understanding incorrect syntax than language speakers, it is necessary to learn the basic syntax of the language being used (in this case, Python). If you leave out a full stop or a comma somewhere in an English sentence, it's likely that people will still be able to understand you. However, if you leave out something in a programming language that is necessary according to its syntax (for example a line break, correct indentation, or a colon), your program will not run.

In Python, line breaks and indentation are used to delimit statements. Consider the If-Else statement in Edgy:



The equivalent Python code is:

```
if a == b:
    foo(bar)
else:
    baz(bar)
```

In Edgy, we can see that the  and  are inside the C-shaped areas of the if statement. When we write this in Python, we need to indent the `foo(bar)` and `baz(bar)` statements accordingly. Here, we've indented them by four spaces, but they can be indented by other amounts (e.g. two spaces, or a tab character) as long as you are consistent.

Take a look at this Edgy script with nested if statements:



If we have nested blocks in Edgy, we have to represent this nesting as indentation in Python:

```
if A == B:
    foo(bar)
else:
    if A > B:
        if A * B > 0:
            foo(bar)
        else:
            baz(bar)
```

Also note that statements in Python generally belong on their own lines:

```
# this works
foo(bar)
baz(bar)

# this doesn't
foo(bar) baz(bar)
```

If you need to split a statement on to multiple lines (for readability), you can use a backslash to denote that the statement continues:

```
# you can write
sum = x + y + z

# as this
sum = x + \
    y + \
    z
```

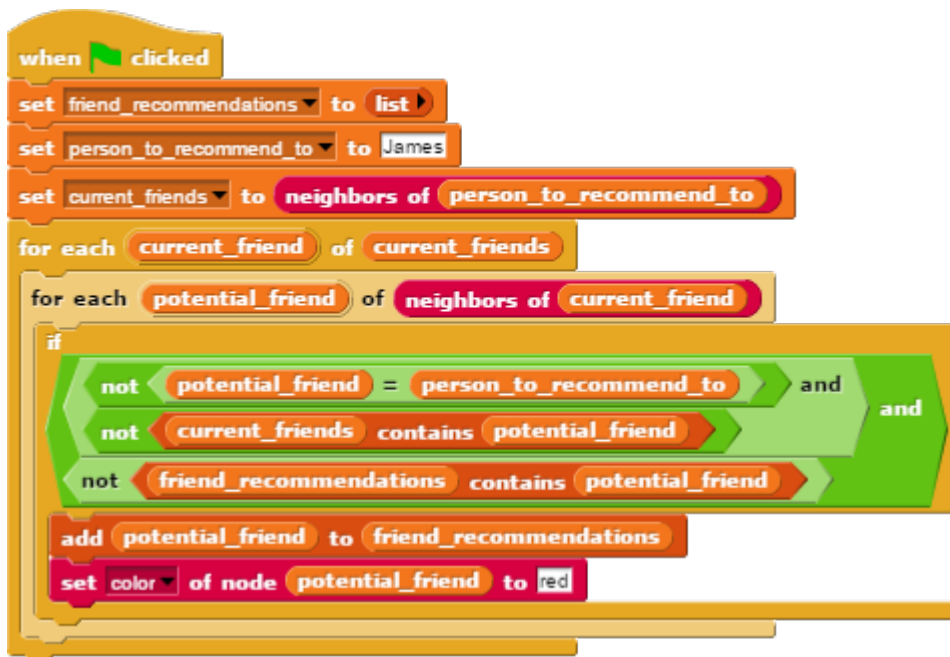
Statements surrounded by brackets can be broken into multiple lines as well:

```
# a long list
list = ['Item 1', 'Item 2',
        'Item 3', 'Item 4']

# a long if statement
if (A == B and C == D and
    E == F and G == H):
    foo(A, B, C, D, E, F, G, H)
```

Making Friend Recommendations in Python

Recall the completed friend recommender from [section 1.6](#):



Let's convert this to Python step by step. We must first start by importing our helper module:

```
from edgy import *
```

We also need to import our graph from our file in our program (since we can't just load it beforehand):

```
# import the graph
G = nx.drawing.nx_pydot.read_dot('friends1.txt')
```

`nx.drawing.nx_pydot.read_dot(file)` is a function that accepts a file name and returns a graph loaded from the file in DOT format. We'll use this to import our DOT format social network from 1.6.

Then we can start the main program with the initialization of `friend_recommendations`, `person_to_recommend_to` and `current_friends`:



- `friend_recommendations` is set to the empty list.
 - In Python, lists are initialized as comma-separated items in square brackets: `['item 1', 'item 2', 'item 3']`. Since we want the empty list, we'll just set `friend_recommendations` to `[]`.
- `person_to_recommend_to` is set to the node, which in this case is the string `'James'`
 - Strings are surrounded by either single quotes (`'string'`) or double quotes (`"string"`)
- `current_friends` is set to the list of all the neighbouring nodes of the person we're recommending to (James)

- We can get the neighbours of a node using the `G.neighbors(node)` function.

So we add to our program:

```
# the main program
friend_recommendations = []
person_to_recommend_to = 'James'
current_friends = G.neighbors(person_to_recommend_to)
```

Then we have some nested for loops:

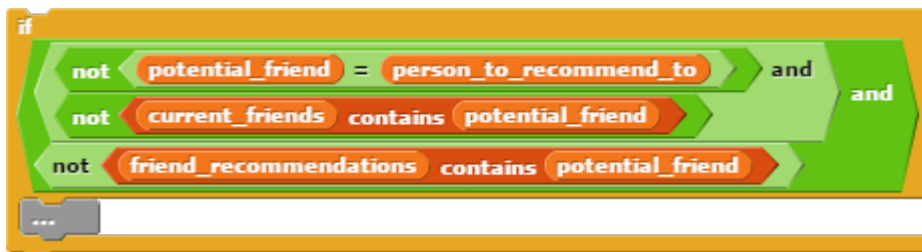


- For each `current_friend` from in the list of `current_friends`
 - For each `potential_friend` in the list of neighbours of the `current_friend`

So we add to our program:

```
for current_friend in current_friends:
    for potential_friend in G.neighbors(current_friend):
        ...
```

Then we have an if-statement:



- If the `potential_friend` is not the `person_to_recommend_to` and
 - The `potential_friend` is not already in the list of `current_friends` and
 - The `potential_friend` is not already in the list of `friend_recommendations`

We can check if an item is in a list by using the `in` keyword:

```
>>> 'A' in ['A', 'B', 'C']
True
>>> 'A' in [1, 2, 3]
False
```

We can also use `not in` to check that an item is not in a list:

```
>>> 'A' not in ['A', 'B', 'C']
False
>>> 'A' not in [1, 2, 3]
True
```

So we replace the ellipses in our inner for-loop with:

```
if (potential_friend != person_to_recommend_to and
    potential_friend not in current_friends and
    potential_friend not in friend_recommendations):
    ...
```

We break this if-statement into multiple lines so that it more closely resembles the Edgy script (and is also easier to read). We also put parentheses around the condition so Python knows it's a single entity. Remember to indent the if-statement to be inside the for-loops.

Then inside the if-statement:



- Add `potential_friend` to the list of `friend_recommendations`
 - The `list.append(item)` function adds an item to the end of a list
- Set the colour of the `potential_friend` to red.
 - Node attributes can be accessed with `G.node[my_node]['my_attribute']`

So we replace the ellipses in our if-statement with:

```
friend_recommendations.append(potential_friend)
G.node[potential_friend]['color'] = 'red'
```

Remember to indent this section to be inside the if-statement.

Then we have to show our results, so add this to the very end of the program:

```
# show the results
print(friend_recommendations)
show(G)
```

Our completed program is:

```
from edgy import *

# import the graph
G = nx.drawing.nx_pydot.read_dot('friends1.txt')

# the main program
friend_recommendations = []
person_to_recommend_to = 'James'
current_friends = G.neighbors(person_to_recommend_to)

for current_friend in current_friends:
    for potential_friend in G.neighbors(current_friend):
        if (potential_friend != person_to_recommend_to and
            potential_friend not in current_friends and
            potential_friend not in friend_recommendations):
            friend_recommendations.append(potential_friend)
```

```
G.node[potential_friend]['color'] = 'red'
```

```
# show the results
print(friend_recommendations)
show(G)
```

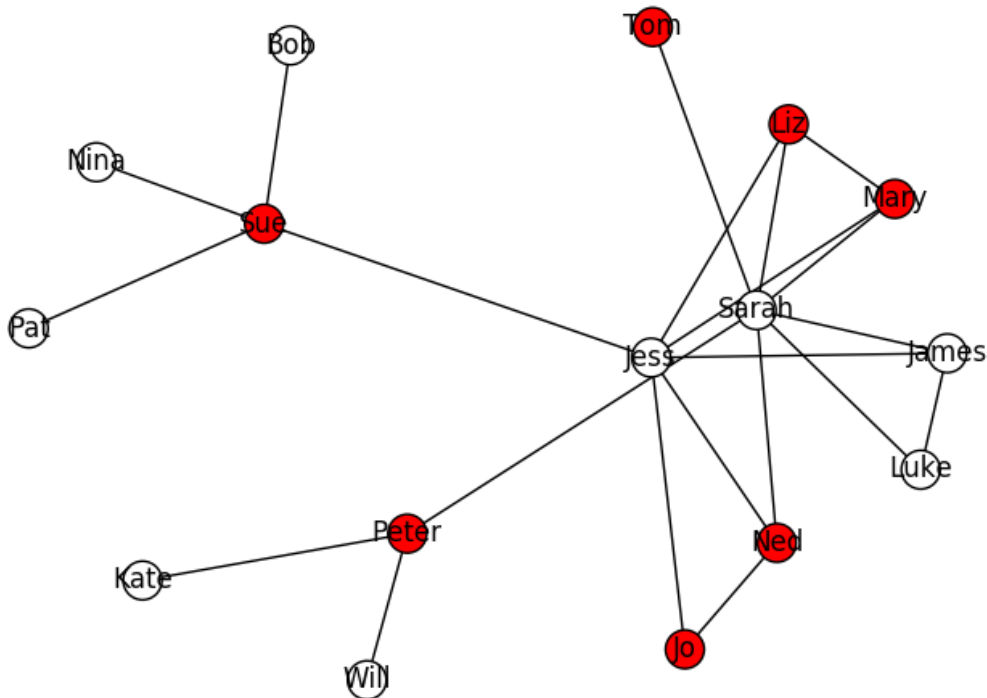
Save this as `friends.py` along with [friends1.txt](#)

(<https://www.alexandriarepository.org/wp-content/uploads/friends1.txt>), open it in IDLE and choose Run > Run Module to run the program. Alternatively, run the command:

```
$ python friends.py
```

This gives the same result as Edgy:

```
['Ned', 'Tom', 'Peter', 'Mary', 'Liz', 'Sue', 'Jo']
```



Custom Block Definitions

In Edgy, it's possible to define custom blocks to perform specific functions:



The equivalent in Python is done using the `def` keyword:


```
def foo(bar):
    print(bar)
```

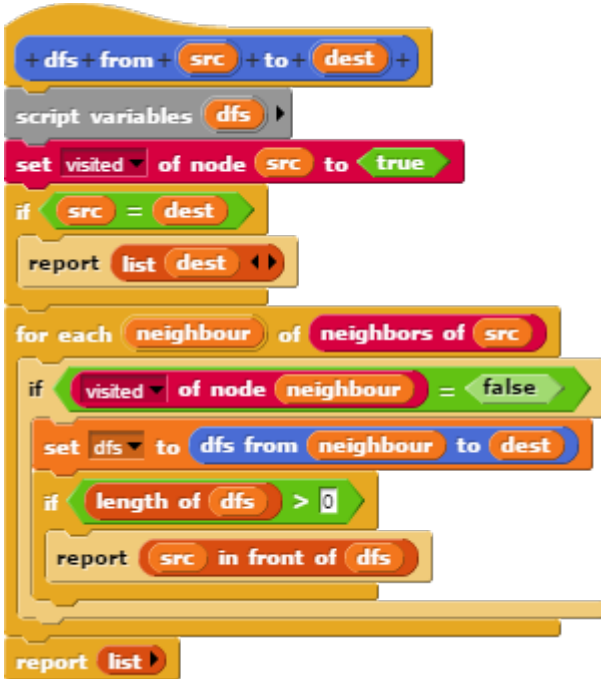
The return keyword is equivalent to the  block in custom reporters:



Can be implemented in Python as:

```
def foo(bar):
    print(bar)
    return(bar)
```

Consider this implementation of depth-first search from the Jealous Husbands example:



In Python, we can write this as:

```
def dfs(G, src, dest):
    G.node[src]['visited'] = True
    if src == dest:
        return [dest]
    for neighbour in G.neighbors(src):
        if not G.node[neighbour]['visited']:
            result = dfs(G, neighbour, dest)
            if len(result) > 0:
                return [src] + result
    return []
```

We've changed the name of the variable `dfs` to `result`, as we've called the function itself `dfs` (not changing the name of the variable will lead to an error due to ambiguity between the variable name and the function name).

Notice that we made `G` a parameter of the function. This allows us to run our `dfs` function on any given graph (instead of just a single global graph) as discussed below.

Using Multiple Graphs

In Edgy, the blocks used for manipulating graphs all tend to work on a single, global graph that we can visualize on the stage. In Python, however, we're able to store graphs in variables just like any other value. So far we've used:

```
>>> G = nx.Graph()
```

This sets the variable `G` to our graph. We can make more graphs if we want:

```
>>> G2 = nx.Graph()
>>> G3 = nx.Graph()
```

And now we can access three independent graphs. If we want to manipulate `G2`, we can do the following:

```
>>> G2.add_node(1)
>>> G2.add_node(2)
>>> G2.add_edge(1, 2)
>>> show(G2)
```

This allows us to use multiple graphs in our algorithms. For example, in an algorithm to generate a minimum spanning tree in Edgy, we may have coloured the nodes that comprised the MST. In Python, we could simply add the relevant nodes and edges to a completely separate graph.

Going back to our `dfs` function, we can, for example easily do a depth-first search on many graphs:

```
G1 = nx.drawing.nx_pydot.read_dot('graph1.dot')
G2 = nx.drawing.nx_pydot.read_dot('graph2.dot')
G3 = nx.drawing.nx_pydot.read_dot('graph3.dot')

dfs(G1, 1, 10)
dfs(G2, 'A', 'Z')
dfs(G3, 'X', 'Y')
```

Using Documentation

While Python is not the most complicated programming language, it would be unreasonable to memorize every little function that ends up being used. Thankfully, documentation exists to help you find the functions you need, and how to use them. It's a good idea to have the documentation open for any libraries you're using when programming, so you don't get stuck trying to 'remember' how to do a specific thing, when you can simply look it up.

Summary

This has been an introduction into moving from Edgy to a conventional programming language. You can see that there is a clear connection between Edgy blocks and Python code, and will now be able to apply ideas and concepts regarding algorithm design to other programming languages as well.

Now that you've had a go at some Python, you might want to read though some other Python tutorials and try to convert some Edgy programs into Python on your own. You can use the reference below to help

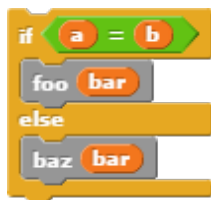
Reference: Translating Edgy to Python

Most of the blocks in Edgy are designed work very similarly to lines of code in languages such as Python. Here is an outline of the conversion of some common Edgy blocks to Python.

Control



```
if a == b:
    foo(bar)
```



```
if a == b:
    foo(bar)
else:
    baz(bar)
```



```
while a != b:
    foo(bar)
```



```
for i in range(1, 10):
    foo(bar)
```



```
for item in collection:
    foo(bar)
```



```
return bar
```

Variables



```
variable = 0
```



```
variable += 1
```



```
collection = ['A', 'B', 'C']
```



```
collection.append("A")
```



```
collection[0]
```

In Python, list indices start at 0 (so the first item is item 0, not item 1).

Operators

Collections

```
collection.insert(0, "A")
```

In Python, list indices start at 0 (so the first item is item 0, not item 1).

```
del collection[0]
```

In Python, list indices start at 0 (so the first item is item 0, not item 1).

```
True
```

```
False
```

```
A < B
```

```
A == B
```

```
A > B
```

```
not predicate
```

```
dictionary = {
    'key 1': 'value 1',
    'key 2': 'value 2'
}
```

Dictionaries can be iterated over with:

```
for key in
dictionary: print key print
dictionary[key]Or:
for key, value in
dictionary.items(): print key print
value
```

```
dictionary['key'] = 'value'
```

```
dictionary['key']
```

```
del dictionary['key']
```

```
'key' in dictionary
```

```
top = stack.pop()
```

Stacks can be implemented as lists in Python:

```
>>> stack = []
>>> stack.append('A')
>>> stack.append('B')
>>> stack.append('C')
>>> stack.pop()
'C'
>>> stack.pop()
'B'
>>> stack.pop()
'A'
```

```
queue.append('item')
```

```
head = queue.popleft()
```

```
pqueue = PriorityQueue()
```

```
pqueue.put((1, 'item'))
```



Looks



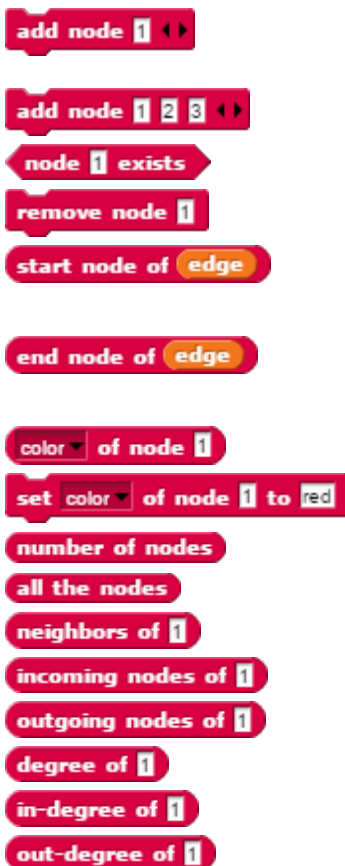
Network



Graph generators can be found [here](https://networkx.readthedocs.io/en/stable/reference/generators.html)

(<https://networkx.readthedocs.io/en/stable/reference/generators.html>).

Nodes



Edges



```
head = pqueue.get()[1]
```

You can retrieve the priority and value of the dequeued element with:

```
priority, value = pqueue.get()
```

```
print('Hello!')
```

```
G = nx.Graph()
```

```
G = nx.DiGraph()
```

```
show(G)
```

```
G.number_of_nodes() == 0
```

```
nx.topological_sort(G)
```

Examples:

```
G_tree = nx.balanced_tree(2, 3)
```

```
G_grid = grid_graph(dim=[2, 3])
```

```
G.add_node(1)
```

You can set node attributes while adding a node:

```
G.add_node(1, color='red')
```

```
G.add_nodes_from([1, 2, 3])
```

```
G.has_node(1)
```

```
G.remove_node(1)
```

```
edge[0]
```

```
edge[1]
```

You can destructure an edge (to obtain the start and end node) using:

```
u, v = edge
```

```
G.node[1]['color']
```

```
G.node[1]['color'] = 'red'
```

```
G.number_of_nodes()
```

```
G.nodes()
```

```
G.neighbors(1)
```

```
G.predecessors(1)
```

```
G.successors(1)
```

```
G.degree(1)
```

```
G.in_degree(1)
```

```
G.out_degree(1)
```

```
G.add_edge(1, 2)
```

```
G.add_edges_from([(1, 2), (3, 4)])
```

```
G.has_edge(1, 2)
```

```
G.number_of_edges()
```

all the edges

label of edge edge 1 2

set label of edge edge 1 2 to value

edges of 1

incoming edges of 1

outgoing edges of 1

Other

Importing a DOT file


 this is a comment /
`G.edges()``G.edge[1][2]['label']``G.edge[1][2]['label'] = 'value'``G.edges(1)``G.in_edges(1)``G.out_edges(1)``G =``nx.drawing.nx_pydot.read_dot('file.dot')``# this is a comment`