Career Explorations -- Networking And Security

Programming Lab 1

Your task, should you choose to accept it, is to figure out a way to pass notes between two people (just like in the pre-lab activity) using the programming language called *Processing*.

We will provide you with some code to get you started and indicate the places where you will need to write your solutions! If any of these steps don't work as expected please raise your hand and one of us will come around to you.

Step 1: **Getting started**

To kick-start the lab, first, you must download the starter code. To do this, Download the .zip file from https://bit.ly/2KBbRUX onto your desktop (the download should start automatically). Then, right click on it and select "Extract". You should see a folder entitled Main on your Desktop.

You may need to download *Processing*. To do this, visit https://bit.ly/2GwpO4L in a browser and the download should start automatically. Once it is downloaded, right click on the file and select "Extract". This may take a few moments.

Once the download completes, open *Processing*. You can do this by clicking on the start menu and typing *Processing* in the search bar. When the program opens, select File -> Open, then click on "Desktop". Select the file Main that you just created and click on Open.

A second Window of Processing should appear. Feel free to close the original one. The processing window should have tabs entitled "Main", "Lab_1", "Lab_2", "Lab_3", etc. If it does not, please flag an instructor over to help.

Select the "Lab_1" tab to start your first programming lab!

Step 2: Explore the Interface

The starter code that we've provided you with will get you started. To run the starter code, press the play button in the top left corner of the screen. A window with 7 stick figures (labelled 0 through 6) should show up. You can click-and-drag to move people around.

Now, press the Run button on the lower right of the window. Nothing should happen, except that the run button should turn yellow and read "Pause".

Sending a Letter. To send a letter from Person 4 to Person 1, click once on Person 4 and then on Person 1. If your program is not paused, you should see a blue envelope stutter across the screen from Person 4 to Person 1. You can do this between any two Persons who are connected by a yellow line (e.g. Person 5 and Person 0, but not Person 2 and Person 3). Play around with sending messages between connected Persons. What happens when you try and send a letter from 5 to 3?

You can toggle the Run/Pause button to pause the animations (try hitting the spacebar , the 'r' key, or the 'p' key). Once you've paused the animation, try clicking the Step button (or similarly the 's' or TAB keys).

At some point, try and send a letter between two people connected by a line (e.g. People 3 and 0) pause the animation before the letter arrives. Then mouseover the letter and notice that the numbers of the sender and the receiver in the black box as **Snd** (for sender) and **Rcv** (for recipient).

If the window freezes, click on the *Processing* window and hit the Play button again.

Step 3: Respond To a Letter

Now it's time to write some code!! The people in our current simulation are rather rude, they receive letters but never respond to them….. Jerks

Your first programming task is to give them some manners. But first, some background.

Code is organized into chunks called *commands*. Some commands are built into the programming language, such as arithmetic addition (+) or subtraction (-). Others are built up by programmers to help organize our thinking.

You will implement the command `itsForMe`, which represents the code that should be executed by a `Person` whenever they receive a letter addressed to them. What do you do when you receive a letter that is addressed to you? You respond to it of course!====

For instance, in the diagram below, Person 1 is about to receive a Letter from Person 4, which is addressed to them (see `Rcv = 1`). On the next step, when they do receive the letter, they will execute the `itsForMe` command on the letter.

What does it mean to "execute a command on the letter"? Examining the `Lab_1` file, we can see the location where you will define the behavior of the `itsForMe` command. It says

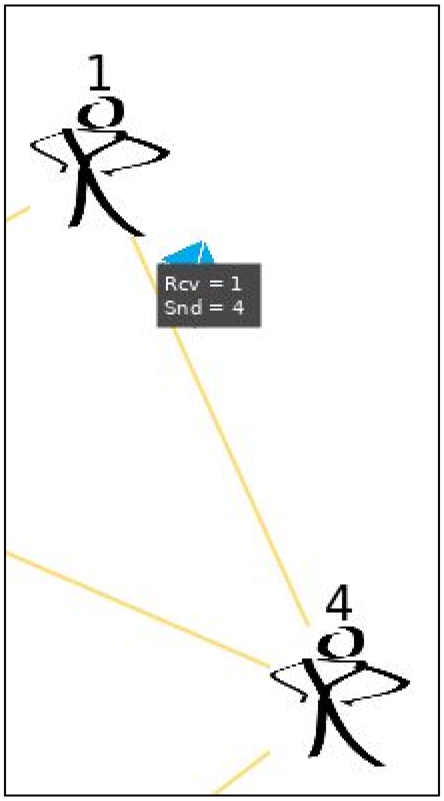
```
void itsForMe(Letter l) {
  // TO DO : fill in code here
}
```

(Beware! Code after // isn't run!)
(don't worry about the `void` for now)

1
itsForMe( )
Rcv = 1
Snd = 4
Rcv = 1
Snd = 4

In this context, the `Letter l` is the letter that the Person needs to respond to. So, in the above example, the `Letter l` will be the letter in the picture. So what should you do? Send a response back to the Person who wrote the letter to you!

Your solution should send letters back and forth between two line-connected Persons. There are a handful of commands that might be helpful in crafting your solution.



`Person getSender(Letter l)`

Gets the sender of the letter `l`.
To execute this command, you type `getSender(l)`, which will behave like a function in math, where it has an input and an output. Here, the input is `l` and the output is the Person who sent `l`. In the example on the right, `getSender(l)` is the Person numbered 4.

`Person getRecipient(Letter l)`

Gets the recipient of the letter `l`. For the letter in the figure to the right, it will return 1.

void `sendALetterTo`(Person `p`)

sendALetterTo will send a letter to the person *p*. For example, in the image on the right, in order to send the letter from Person 4 to Person 1, Person 4 had to execute `sendALetterTo`(`p`) where `p` was equal to the Person numbered 1.

void `replyTo`(Letter `l`)

Using this function will look at the sender of the letter `l` and send a letter back. In the example on the upper right, if Person 1 executes `replyTo`(`l`) where `l` is the depicted letter, it will send a letter back to Person 4.

Step 4: Send A Far-away Message

You may have noticed that if you try to send a message to a person to whom you are not connected by a yellow line, nothing happens. This is because you currently "don't know" how to get there. We will say that if two Persons are connected by a single yellow line, they are *neighbors*.

Think back to the group activity we just did. How did you pass a note to someone who wasn't sitting next to you? You probably instructed a person who was sitting next to you to give it to your far-away correspondent.

Conversely, what happened when you received a note that was destined for a person who you didn't know how to get to? What about when you did know?

We've designed the commands to mirror this way of thinking. There is one command that will be executed when you `doKnow` where to forward the Letter to, and there is another command that will be executed when you `dontKnow` who to give the letter to. We've gone ahead and filled in the code for the `doKnow` command. **You should only fill in the `dontKnow` command for now.**

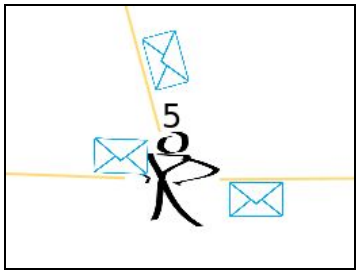
Depending on your solution, you may see many white letters. This is okay! These letters simply represent the fact that the sender doesn't know how to get to the recipient and is exploring its neighbors. In fact, if you successfully implement a far-away communication, you should see a checkmark at the recipient Person and may see many sad faces all over the place.

Your ultimate goal is to be able to (after a period of chaotic white-letter exploration) communicate with blue letters, which means learning information about the network -- we'll see how to do this in step 5. This will also fix the sad faces.

You may find the following commands (in addition to the previous ones) helpful in crafting your solution:

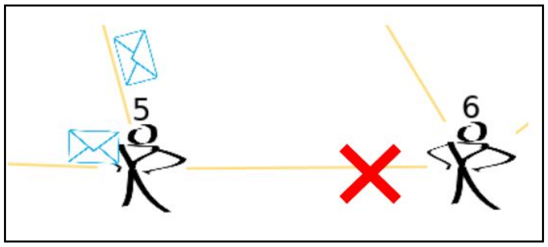
```
void forwardToAll(Letter l)
```

The *forwardToAll* command will forward a letter `l` to every neighbor. For example in the image to the right, executing this command on Person 5 will send a letter to each of its three neighbors.



```
void forwardToAllExcept(Person neighbor, Letter l)
```

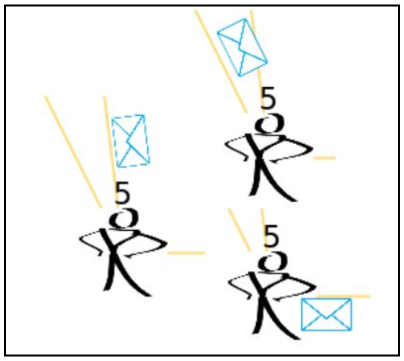
The *forwardToAllExcept* command will forward a letter `l` to every neighbor except the `neighbor` given as an input. For example, if this command is



called on Person 5 and neighbor is Person 6, then Person 6 will forward to only its other neighbors.

`Person getRandomNeighbor()`

getRandomNeighbor will return a random neighbor. On the right we show what happens when Person 5 executes `forwardTo(getRandomNeighbor(), l)` three different times for some Letter `l`.



`Person getRandomNeighborExcept(Person neighbor)`

getRandomNeighborExcept will return a random neighbor except the `neighbor` given as input. In the example on the right, calling this command on Person 5 with neighbor equal to Person 6, will permit all of the options except for the bottom right situation.

`void forwardTo(Person neighbor, Letter l)`

This function will take in a *neighbor* and pass along the letter `l` to them. In the example above. Person 5 is forwarding to a router provided by the function `getRandomNeighbor()`. Note that the `forwardTo` function will preserve the color of letters.

`boolean isMyNumber(int number)`

Each person has a number. *isMyNumber* will determine if the current person's number is the input *number*. If so, it will return true, otherwise it will return false. For example on Person 5, `isMyNumber(5)` is `true`.

Step 5: Learning How to Deliver Letters

It seems silly to have to explore the network the way that we did in Step 5 every time we want to send a Letter across the room. What if we could use the information that people observe to help us save the effort of sending so many extra letters through the network? In this step, you will implement the command `discover(Person neighbor, Letter l)` which is called whenever a Person receives a Letter `l` from their `neighbor`.

Lets Personify. What fact(s) can you learn if Jonathan hands you a letter that is from Brianna and addressed to Eric?

You know that Jonathan can return a letter to Brianna

So, who should you hand your letter to if you want to send or forward a letter to Brianna?

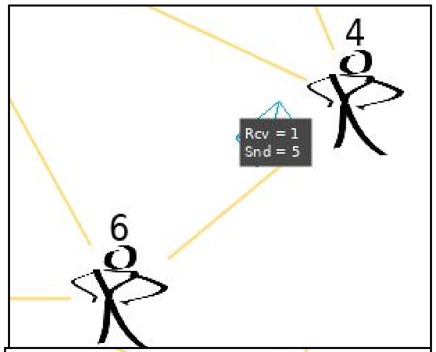
To Jonathan!

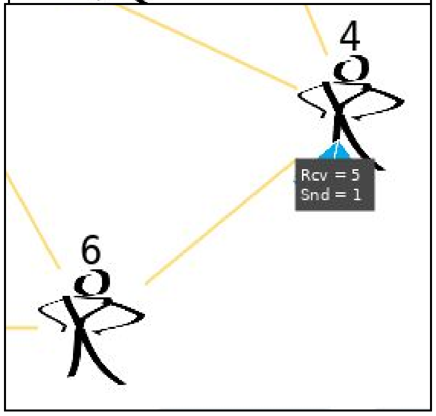
All you need to do is implement this reasoning in the `discover` command! You may find the following commands useful in crafting your solution, and you may get inspiration from the code in the `doKnow` method.

`void memorizeHowToGetTo(Person p, Person neighbor)`

`memorizeHowToGetTo` makes you "know" that to send a letter to Person `p`, you can just give it to `neighbor`. This is closely related to the `rememberHowToGetTo` method used in the `doKnow` command definition.

For example, in the image to the right, Person 4 was given a letter (by Person 6) addressed to Person 1 sent by Person 5. Person 4 can execute `memorizeHowToGetTo(p,neighbor)`, where `p` is Person 5 and `neighbor` is Person 6. From this point on, Person 4 will hand future Letters that are addressed to Person 5 to Person 6 (as in the below right).





`Person rememberHowToGetTo(Person p)`

`rememberHowToGetTo` returns the neighbor who can deliver letters to Person `p`. In the figure to the right (described above), If Person 4 executes `rememberHowToGetTo(Person p)` where `p` is Person 5, it returns Person 6.

Extra Credit Task 1: Experiment with different routing strategies

If your solution to Step 4 was random, try forwarding to all of your neighbors, or only to certain ones. If you forwarded to all of your neighbors, try a random solution! Play around with these different commands and figure out which solution seems the "best". Call one of us over and explain your reasoning.

Extra Credit Task 2: Static Routing

The solution so far is a *dynamic* routing scheme, in that it figures out how to forward letters based on what happens when the program runs. Try and implement a static routing scheme. Here there is no distinction between knowing and not knowing, because you always know where to hand the letter. To do this, you need to write special rules for each person. You will need if-statements and the `isMyNumber` function to figure out where to forward letters.

Hint: Each Person will forward all of the Letters it receives to the same neighbor. For instance, Person 3 may always forward to Person 4.