Debugging

• warm-up activity

• find a partner

• find the bugs (errors) in the Scratch code

• it is okay if you don’t come up with the “Scratch” solution but you should talk with you partner about the bug and how each of you might fix it

• hint: there is no one right answer!
Possible Scratch Solutions
What process did you follow when looking for bugs?
• term coined by **Grace Hopper** in the 1940s when she found a moth in the Mark II computer she was working on

• the moth was jammed in one of the relays causing the computer to fail

• **debugging**: method of figuring out why a process or system doesn’t work properly.
Debugging

• will be talking about 2 overall themes throughout the lecture

• debugging software/hardware you use in everyday life but do not write or build

• debugging code that you write (e.g., Processing code)
Who is to blame for bugs in software?

- You! .. usually
- computers are predictable
- they do exactly what they are told to do
- computers can’t think for themselves
- if they weren’t programmed to do it; they will fail
- software errors are much more rare than human errors!
Consider 0’s (zeros) and O’s (oh’s)

- or even l’s (el’s) and 1’s (ones) for that matter

- what if the user types in an oh when they mean a zero

- how could the computer be programmed to figure it out?
Consider 0’s (zeros) and O’s (oh’s)

• could be programmed to converted oh’s to zero’s

• but what if each has a different meaning

• can you think of any situations where it would be important to know which one the user entered?

• now it becomes very important for the user to be very precise when entering information
Debugging in Everyday Life

• you more than likely have debugged long before the start of learning Processing in this class

• can you think of anything you have “debugged”? 
Debugging Process

• 1. run the program; reproduce the same exact error

• 2. state/determine the error

• 3. eliminate the “obvious” cases

• 4. Divide! Eliminate the parts of code that work properly

• 5. Go back to step 1
Debugging Process

• 1. run the program; reproduce the same exact error

• why?
Debugging Process

• 2. state/determine the error

• 3. eliminate the “obvious” cases

• how do we do this in Processing? in Scratch?
Debugging Process

• 4. Divide! Eliminate the parts of code that work properly

• why would this make it easier?

• let’s take a look at Chelsey and Lisa’s code

• they have a bug, but their code is 7 pages long! We need to narrow down the problem.
Debugging Process

• in groups of 3 or 4 find the bug in their code

• fix it on paper

• Chelsey & Lisa prepare a brief explanation to give to the class about the bug once they have found it
Debugging Process

• 5. Go back to step 1

• why?

• When should you stop the process?
User Friendly Programing

• how can you write user friendly code so that the user knows when they have entered “bad” information?

• lets take a look at Shari and Stefan’s code