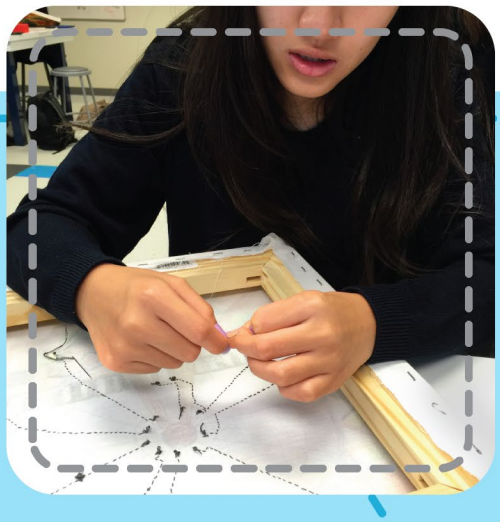
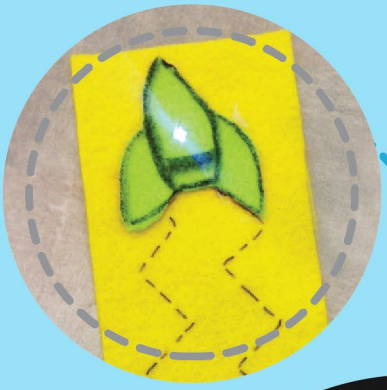
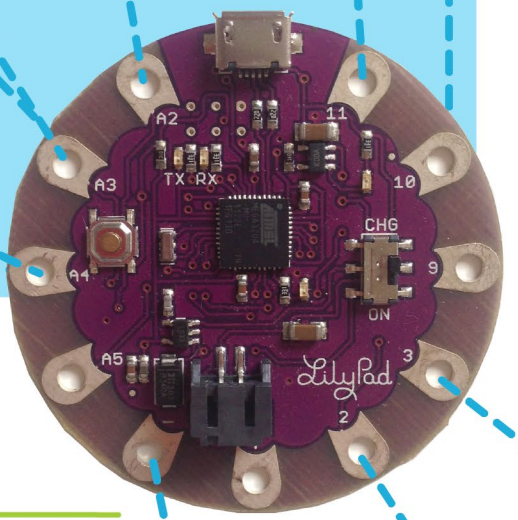
E-textiles

Technical Guide







This guide is a companion resource to the curriculum unit and professional development workshops for the electronic textiles activities in *Exploring Computer Science* (www.exploringcs.org). Tomoko Nakajima, Janell Amely, Deborah Fields, and John Landa developed this guide together with Yasmin Kafai, Joanna Goode, Gail Chapman, John Ottina, Pamela Amaya, and Jane Margolis as part of a National Science Foundation grant (1027736).

Please cite this work as:

Fields, D. A., Nakajima, T., Amely, J., Fields, D., Landa, J., Amaya, P., & Ottina, J. (2018). *Stitching the Loop: A Resource Guide for using Electronic Textiles in Exploring Computer Science*. Exploring Computer Science. Available at http://exploringcs.org.

Table of Contents

OVERVIEW

This resource guide provides teachers with information about concepts, common issues, and instructions to help students learn to sew, design circuits, and write computer programs to make their own electronic textiles, and gives more background information, explanations and step-by-step instructions than are included in the ECS e-textiles curriculum lesson plans.

Electronic textiles (shorter: E-textiles) are wearables embedded with microcontrollers, sensors, and lights. Designers of e-textiles use materials like conductive thread, conductive fabric, and flexible circuit boards. E-textiles are used in many different contexts from education to sports, fashion, military and medicine. Examples of e-textiles include astronaut space suits, wearable medical devices, haute couture fashions, entertainers' costumes, and athletic-wear.

1 Basics: This section includes the topics circuits, crafting and coding.

2 Troubleshooting: You will make a few mistakes as you make electronic textiles—everyone does! In this section we describe various common errors that come up when crafting and programming electronic textiles, identify possible sources, and then suggest strategies on how to fix them.

3 Materials: This section includes descriptions and images of many of the materials used in electronic textile projects.

4 Ideas & Inspiration: Sometimes coming up with a cool project idea is the hardest part! Before you get started, look through photos and videos of example projects made by other people. This section also includes instructions on how to make your own sensors.

5 Glossary: This section contains definitions for many terms you will encounter in the curriculum or this manual.

TOC

Examples of Electronic Textiles

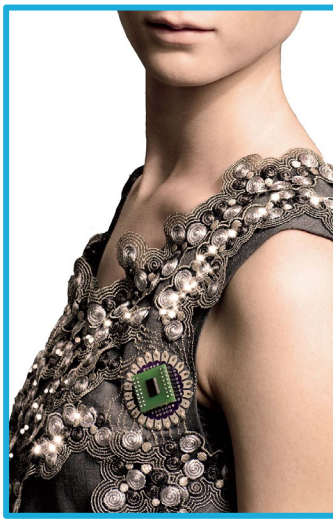
The Music Jacket by Maggie Orth, Emily Cooper, Rehmi Post: Jacket incorporates an embroidered keypad, mini MIDI synthesizer, speakers, and custom music software.

Turn Signal Jacket by Leah Buechley: Switches on the right and left sleeves of the jacket control the LEDs, activates the "turn signals" when pressed.

Know-It-All-Knitting-Bag by Kalani Craig: LEDs are positioned on the bag to reflect traditional knitting charts, and the display blink patterns indicate the knitting pattern used to create the bag.

TOC

The Climate Dress by Diffus Design: Light patterns on the dress change based on CO2 levels in the air. Microcontrollers and CO2 sensors are connected through complex embroidery with conductive thread.



Kit of No Parts by Hannah Perner-Wilson & Leah Buechley: DIY textile-based sensors can be made with accessible materials, and used to create e-textiles projects conductive thread.

TOC

Ethical and Safety Issues

E-textiles merge arts and crafts with electronics and computers. The use of technology comes with some ethical imperatives. The Association for Computing Machinery[1] published this list of important and agreed-upon general ethic principles for working with computers:

- Contribute to society and to human well-being, acknowledging that all people are stakeholders in computing.
- Avoid harm.
- Be honest and trustworthy.
- Be fair and take action not to discriminate.
- Respect the work required to produce new ideas, inventions, creative works, and computing artifacts.
- Respect privacy.
- Honor confidentiality.

Adding "power" through circuitry to do-it-yourself projects presents its own challenges and responsibilities. One issue is the proper storage and disposal of our electronic materials.

Batteries: Coin cell batteries should be stored so that they're not touching one another to prevent short circuits. Lithium primary batteries provide extremely high currents and can discharge very rapidly when short-circuited, resulting in overheating, rupture, and even explosion. This is why some lithium-based batteries shipped in high quantities can sometimes qualify as a Class 9 hazardous material. Try to store batteries separate from projects, to lock the batteries away after use, and to recycle "dead" batteries at appropriate recycling centers (some do not accept lithium primary batteries). Some stores that only sell batteries will take the used coin cell batteries to recycle. Regulations for disposal of batteries vary widely; local governments may have additional requirements over those of national regulations. **Do not dispose of them in the trash!** Heavy equipment at landfills can easily crush battery cases and exposed lithium can cause slow-burning landfill fires and react violently to water. [2]

Sharp objects: Many schools and school districts, including LAUSD, have zero tolerance policies for possession of sharp objects on campus. This ban might include sewing needles, seam rippers, fabric scissors, Xacto knives, etc. that are regularly used in e-textiles classrooms. Be sure to obtain

[1] Association for Computing Machinery. (2018). *ACM Code of Ethics and Professional Conduct*. Retrieved from ethics.acm.org.
[2] Buchmann, I. (2016). Learn about batteries. *Battery University.* Retrieved from http://batteryuniversity.com/learn/article/primary_batteries.

TOC

proper authorization for use of these items on school grounds before providing them to students. This might include obtaining assent and consent from students and their guardians, and also keeping the site administrators aware, too. Make sure the items are stored in locked areas when not in use and that first aid kits are nearby for accidental nicks and cuts. Think about starting a "check-out" program to keep track of all sharp objects during work time, and also make sure that at the end of that time, all sharp items are returned.

Glues: Though not hazardous through general classroom use, crafters' glues and solvents have been used by drug abusers ("glue sniffing") as highly dangerous intoxicative inhalants.[3] Check http://bit.ly/gluesafety to be aware of potential hazardous ingredients before sharing them with students.

[3] Medical Online. (2013). Glue sniffing. *IRG.* Retrieved from www.medicalonline.com.au/medical/drugs/glue_sniffing.htm.

TOC

BASICS: CIRCUITS, CRAFTING, CODING

This section includes tips for beginning crafters, images that illustrate difficult to explain concepts, and a list of common pitfalls to avoid.

Circuitry

Before we dive into making electronic textile projects, we have to understand how the electronic elements are powered. In order for something to be electronic, it must have a functioning circuit with some basic components: A power source, a conductor, and a load. [4]

1. Electricity

Batteries and power outlets provide electricity, and each power supply has a certain number of **volts** that it provides. A volt is a measurement of the electrical potential produced by the battery, or the utility grid that delivers electricity to the outlet in our walls. This voltage is available for our use, but in order for the electricity to power anything, the energy has to be able to move. The electricity has to follow a path (circuit) to create a **current**.

2. Current

Every source of electricity has two sides, and we need a voltage difference in order to get electricity to flow. Electricity flows from an area of higher voltage to an area with lower voltage. If we create a circuit between something with higher voltage and something with lower voltage, the **current** will flow between them. This is why batteries have two different metal ends and outlets have at least two holes. In batteries and other **DC** (direct current) voltage sources, these sides (**terminals**) are either **positive** (+), or **negative** (-). The positive terminal has a higher voltage than the negative side. When we measure voltage, we usually say that the negative is 0 volts, and the positive terminal is represented by however many volts it supplies. A confusing fact: Current flows from positive to negative, but the electrons flow from negative to positive. This circular path, or **circuit**, starts and ends at the power source, and is always required to get electricity to flow.

3. Conductors

Electrical energy can only flow through materials that can conduct electricity, such as metal and certain kinds of liquids. When we give electricity a path to follow from the positive to the negative terminal of the power source, the energy will move through this **conductor**. If a conductor does *not* make a solid connection from the positive to the negative, the electricity has nowhere to go and will not move.

---

[4] Grusin, M. (n.d.). What is a circuit? *Sparkfun Electronics.* Retrieved from https://learn.sparkfun.com/tutorials/what-is-a-circuit.

TOC

Current is LAZY. We have to give it somewhere to go and something to do, or it won't be motivated to do anything.

> **WARNING:** Human bodies are also conductive, as we will learn in the e-textiles Human Sensor Project. This is why we are told *not* to stick metal forks into outlets or use electrically powered appliances near water. For the most part, the projects in this unit are very low powered – only 3.3-5 volts – and will not shock anyone. Still, caution is always warranted.

4. Loads

We build circuits to make electricity function, to do useful things for us. We do this by connecting items (components) between the positive and negative terminals. By creating a circuit, powered with the appropriate voltage of electrical current, the electricity can make light, make noise, sense light or dark, These components are called loads, because they "load down" the power supply, just like a person is "loaded down" when carrying something. If we connect the positive side of a voltage source to a load, like a Light Emitting Diode (LED) that requires electricity to function, then connect that load to the negative side of the voltage source, that LED should turn light up.

> **WARNING:** When connecting the positive to the negative side of a power supply, *without* giving it a load, it creates a **short circuit**. This will drain the battery. In rare occasions, batteries might even explode, though it's more likely that the area of the short circuit will get hot and possibly burn the textile material or you. Find more information about short circuits in the Electrical Problems section.

Example of a simple circuit

Figure 1: Incomplete circuit

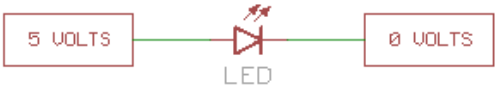


Figure 2: Complete circuit
Images from www.sparkfun.com, CC BY-NC-SA 3.0.

See the difference between Figure 1 and Figure 2? In Figure 1, the LED is dark (doesn't turn on) because the conductor on its right side is not connected to the negative side of the power source. Remember that the negative end of a power source is sometimes indicated as 0 volts because the voltage is "used up" when it flows through the load, and because the electrons leave from the negative terminal and

TOC

travel to the positive terminal. Here is another way to diagram this complete simple circuit (how to draw a circuit diagram is explained later in this Technical Guide):

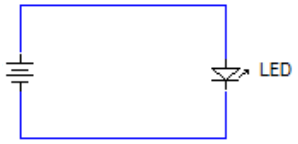


Figure 3: Simple circuit diagram
Image from www.mycircuits9.com.

5. Resistors

It is possible to load down a circuit too little—so that too much energy is flowing through our components. This will eventually burn out our components or even the power supply. **Resistors** are components, which have a specific, never-changing electrical resistance that limits the flow of electrons through a circuit. They are passive, meaning they only consume power and can't generate it. Resistors are added to circuits where the current is too strong for the load. This electrical resistance is measured in **ohms**. In Figure 4, a resistor is necessary because a AA battery produces too much energy for one LED to consume. The resistor is limiting the amount of energy that the LED has access to, so that it does not burn out quickly.

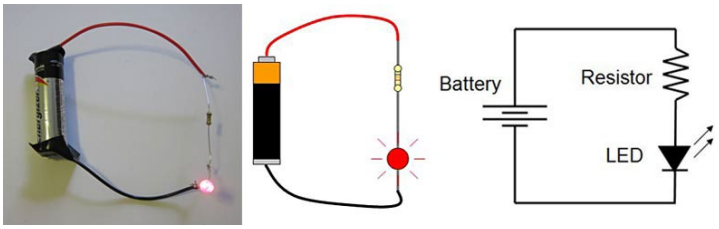


Figure 4: Three depictions of a simple circuit with an added resistor
Image from www.sparkfun.com, CC BY-NC-SA 3.0.

In electronic textiles, the materials we use are carefully designed and ideal for the voltage necessary for our projects. It is possible to complete all of the activities in the curriculum with one-cell batteries and without using resistors. If we were to use stronger batteries, conductors that have less resistance than conductive thread (like steel wires), or if we were to plug the project into a wall outlet, we would need to measure the voltage and add the correct resistors.

6. Switches (open and closed circuits)

A **switch** can be placed in a circuit to shut off or turn on different functions by controlling the current. It is a critical component in any electronic system that requires user interaction or control. Accidentally letting too much current flow through part of our circuit (because the load is too light) can cause a fire. Most power supplies have a safety mechanism built into them to break the circuit if it detects too much current. A **circuit breaker** is a kind of a switch that opens the circuit to stop electricity flow. This

TOC

is the reason all homes and buildings have circuit breakers to prevent overloaded electrical components from starting a fire.

Figure 5: Different switches used in electronic textiles
Images from www.sparkfun.com, CC BY-NC-SA 3.0.

A switch can only exist in two states: off or on. When the switch is off, it creates an opening or a gap in the circuit, preventing the current from flowing. In the on state, a switch acts as a conductor by closing the circuit. Go to http://bit.ly/falstadcircuit to interact with a circuit that has a switch, or turn on and off the light switch of the room we are in.

7. Sensors

Like a switch, a sensor also generates input signals. But sensors can "read" (or "sense") a range of input conditions, and these readings can be read by the computer and used to set different outputs depending on the inputs. For instance, a motion sensor or smoke detector produce a particular signal when motion or smoke are detected. When the conditions change in the sensor's coverage area, the system responds according to the pre-programmed parameters (the motion detector light turns on, or the smoke alarm goes off).

Figure 6: Devices that use sensors—a metal detector, an MRI machine, and safety airbags on a car
Images from www.smokymountainhobbies.net, www.thehindu.com, & www.carsdirect.com.

In this e-textiles unit, we introduce a tactile sensor that captures and records different levels of physical touch from the user or operator. Many of us interact with tactile sensors every day with our touchscreen electronic devices, computer mouse, or laptop scroll-pad.[5] Unlike these high-tech machines, our touch sensor is made with simple household items: aluminum foil, adhesive, and an iron. Most of the other sensors we use are already installed on the Circuit Playground. However, if you

[5] Techopedia. (2016). Touch sensor. *Techopedia, Inc.* Retrieved from www.techopedia.com/definition/30234/touch-sensor.

TOC

used a different microcontroller, you may need to purchase additional sensors and switches. Below are examples of LilyPad sensors that may be purchased online for use in e-textiles projects.

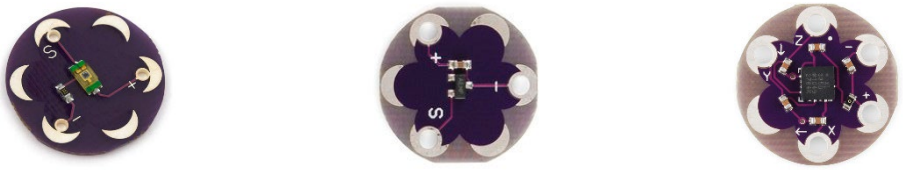


Figure 7: From left to right, Lily Pad light sensor, temperature sensor, and motion sensor (accelerometer)
Images from www.patsyweartech.wordpress.com.

8. Other Output (buzzers, .mp3 Speakers, etc.)

Like in the Lego Mindstorm kits, there are a variety of components other than LEDs that can be sewn into our projects. Output components like mp3 speakers may be purchased from Sparkfun, Adafruit or other sellers to be programed by a Circuit Playground microcontroller. The Circuit Playground also comes with a buzzer (produced pitched sounds).

9. Drawing Circuit Diagrams

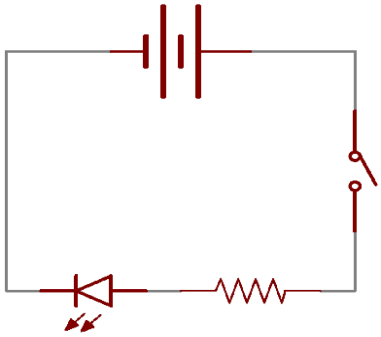


Figure 8: A circuit diagram with a two-cell power source, an LED, a resistor, and a switch. When the switch is closed, current flows and the LED illuminates. Otherwise the current flow is impeded, and the LED receives no power.
Image from www.sparkfun.com, CC BY-NC-SA 3.0.

Circuit diagrams are very helpful in guiding us as we juggle different components and materials. Like "blueprints" for builders, electricians, auto technicians, hardware designers, and electrical engineers, in e-textiles circuit diagrams are commonly used to communicate plans. To learn some of the basic **schematics** (symbols) of circuitry, watch this 6-minute YouTube video: http://bit.ly/simple-how-to. Also, visit this page at Sparkfun for an index of common electronic schematics: http://bit.ly/sparkfunschematic.

Every project in our e-textiles curriculum requires a circuit diagram, meaning that we should make a plan before creating anything. Some of the diagrams use different conventions, but each diagram should have some depiction of the following elements:

- Power source
- Loads (LEDs, buzzers, motors, resistors, etc.)
- Connector lines (circuit traces, including sewn conductive thread)

TOC

- Positive and negative charges clearly marked at every connection
- Any switches or sensors that are also incorporated

As long as we clearly mark these elements so that everyone can understand them, we can choose whichever schematics are the easiest to remember.

NOTE: Marking the positive and negative side for every component in a circuit diagram has been a lifesaver! It is very easy to place a component backward, as indicated in the Troubleshooting Circuitry and Crafting Guide. We all have memories of sewing a beautiful project, only to have to rip out stitches to turn a component around.

10. Power Source

For all of our e-textiles projects, we will use a 3-volt, lithium button battery (CR2032), which has a positive charge on one side and a negative charge on the other (unmarked) side. This one-cell battery is commonly drawn in circuit diagrams with two parallel lines (Figure 10). The longer line indicates the positive charge and the negative charge is demonstrated by the shorter line.

Figure 9: An image of a coin cell battery
Image from www.sparkfun.com, CC BY-NC-SA 3.0

Figure 10: The schematic for a coin cell battery
Image by Tomoko Nakajima.

WARNING: Putting all the small components (batteries, needles, LEDs) into a bag together will **cause short circuits** and at best will only drain the battery. Keep batteries separate from other components so they do not accidently create a circuit in a pocket or backpack. A neat trick is to tape them individually onto a stiff piece of paper or a bag.

After the first e-textiles project, we introduce a sewable **battery holder**. This component has no charge on its own, but it is designed to hold the cell battery in place. They may have multiple **pins** (holes for stitching) that we can sew through with conductive thread so that the battery can power our circuit. Because of this, we may want to use a modified schematic to visualize where to connect the positive and negative of our circuit.

TOC

Figure 11: Two types of sewable battery holders
Images from www.sparkfun.com, CC BY-NC-SA 3.0 & www.kitronik.co.uk.

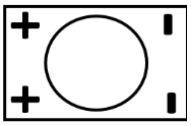


Figure 12: The modified schematic of a battery holder
Image by Tomoko Nakajima.

NOTE: In later e-textile projects that use Circuit Playgrounds, the microcomputer comes with a rechargeable power source. This battery does carry both positive and negative charges (as evidenced by the red and black-cased wires) but it is not necessary to indicate this battery in our circuit diagrams, since it plugs directly into the Circuit Playground (Figure 14).

Figure 13: LiPo battery
Image from www.sparkfun.com, CC BY-NC-SA 3.0

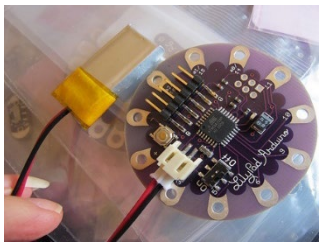


Figure 14: A LilyPad microcomputer (similar to a Circuit Playground) with the rechargeable battery plugged in
Image from www.flickr.com: Rain Rabbit.

In general, we will be only using light-emitting diodes (LEDs) as loads for our projects, and we will primarily be using the following two LED options, which are both sewable (Figure 15 and Figure 16). In schematics, lights can be drawn different ways (Figure 17 and Figure 18). Choose the one that is preferred and use it consistently.

Figure 15: The LilyPad LED
Image from www.sparkfun.com, CC BY-NC-SA 3.0.

Figure 16: Wired LED
Image from www.conrad.com.

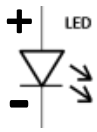


Figure 17: Traditional schematic for an LED
Image from www.kitronik.co.uk.

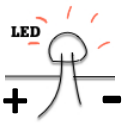


Figure 18: Modified schematic for e-textiles
Image from www.sciencefriday.com.

TOC

11. Connectors (interchangeable with Conductors, Lines, or Traces)

The **connectors** or the lines of our circuit can be drawn with solid lines, with positive lines in **red** and negative lines in **black**. These are the same colors used in electrical wiring. Alternatively, we may also simply indicate the charges clearly with (**+**) and (**-**) symbols (Figure 19).

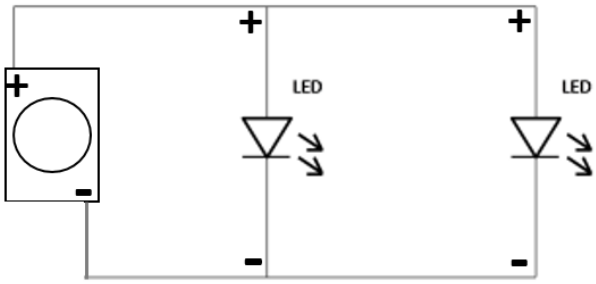


Figure 19: e-textiles circuit diagram with modified schematics
Image by Tomoko Nakajima.

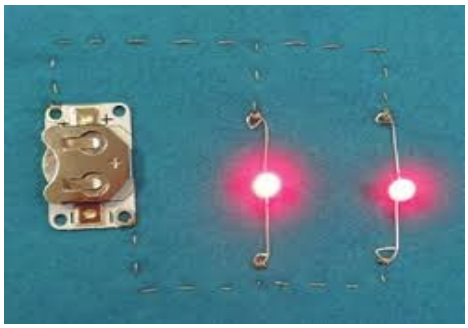


Figure 20: Photo of the actual project
Image from www.kitronik.co.uk.

12. Switches

Switches in a circuit are drawn very simply as a diagonal line. It represents an opening in the circuit, which can be closed when the user wants the circuit to function. Switches can be placed on a negative or a positive line, as they do not require a particular polarity.

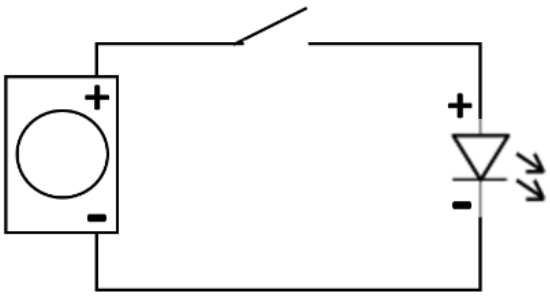


Figure 21: e-textiles circuit diagram with a switch and modified schematics
Image by Tomoko Nakajima.

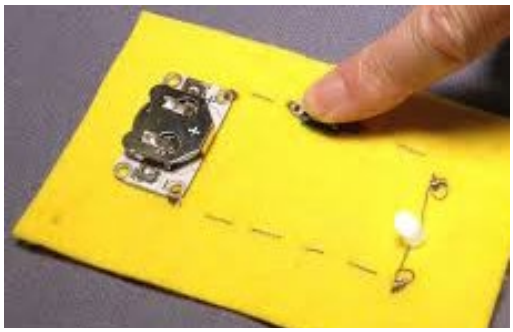


Figure 22: Photo of the actual project
Image from www.gazetteunion.com.

TOC

13. Sensors

Sensors do not have a set schematic or symbol, as they vary greatly in shape, size, and function. The most important thing is to label the sensor accurately. Like switches, sensors are also not negative or positive, though the sensor must be connected to something that will program the sensor and read the range of input received from the sensor (like an analog pin on the Circuit Playground).

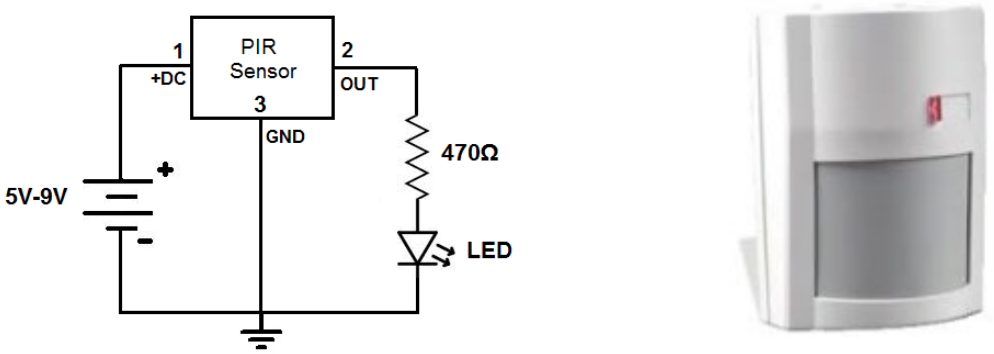


Figure 23: Now we're really getting the hang of it! Can we understand this diagram for a motion detector?
Images from www.learningaboutelectronics.com.

TOC

Sewing

Whenever we are making things by hand, make sure that our workspace is clear of liquids or sharp objects, anything that might damage the work. Also keep hands clean and avoid handling food or drinks while crafting, because some components contain lead or nickel. It is a good idea to wash our hands when we are done, as well.

1. Conductive thread

Conductive thread is thicker than standard sewing thread. It also has a fuzzy texture that frays and tangles easily (Figure 24).[6] This YouTube video explains the properties of conductive thread: http://bit.ly/cond-thread.

Figure 24: Close-up photos of conductive thread
Images from www.solderingsunday.com & www.etextilelounge.com.

Preparing conductive thread for hand-sewing

Step 1: Cut a piece of conductive thread, about two feet long. When the thread is too long, it might get tangled as we sew. If it's too short, we will have to tie off and re-thread more often.

Step 2: Pass the cut thread through the beeswax twice to smooth down the strands (Figure 25).

Step 3: Pinch the end of the thread flat and push it through the eye of the needle. Then pull the loose end through from the other side.

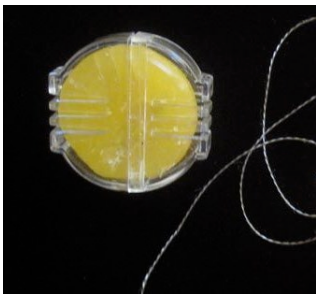


Figure 25: Conductive thread and beeswax
Image from www.instructables.com: Lynne Bruning.

NOTE: If the thread is fraying as we sew, use more wax, water, or saliva to smooth it out.

NOTE: There are different sizes of needles. Thicker needles like the tapestry needles that we provide are easier to thread but do not move through tightly woven fabric as smoothly. The needles that are

[6] Peppler, K., Gresalfi, M., Tekinbas, K. S., Santo, R., & Buechley, L. (2014). *Soft circuits: Crafting e-fashion with DIY electronics*. (pp. 61-62). Cambridge, MA: MIT Press.

TOC

sharper tend to have smaller eyes that are more difficult to thread. See this YouTube tutorial for choosing special needles for our projects: http://bit.ly/chooseneedle.

2. Make a starting knot (single thread)

After the needle has been threaded, tie a standard knot for hand sewing on one end of the thread.[7] Here are step-by-step instructions, photos, and a quick YouTube video: http://bit.ly/sewingknot.

Making a starting knot:

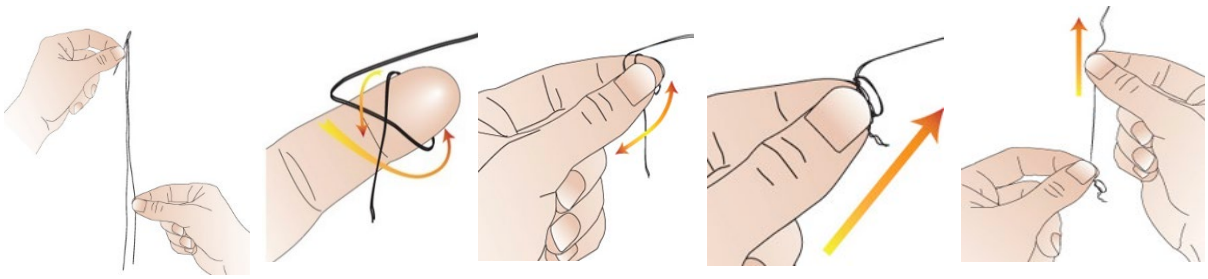


Figure 26: Making a starting knot
Images from www.mcnett.com.

Step 1. Pull one end of the thread so that it's longer

Step 2. Wrap the thread around the tip of the index finger twice

Step 3. Use the thumb to pinch the X where the thread loops cross

Step 4. Roll the loops a few times, and then slip the twisted thread off the finger. The loop should look a little messy. Don't worry!

Step 5. Pinch the tangled loop and pull the thread to tighten. The loop should pull together into a knot. This might take some practice!

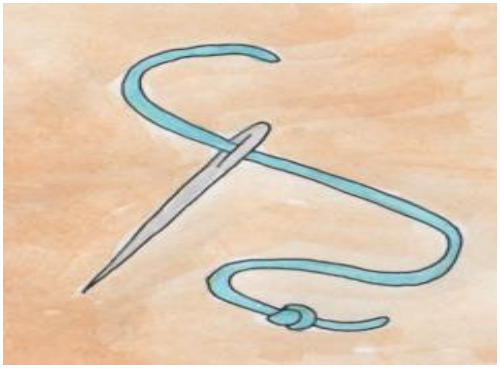


Figure 27: Our single-threaded needle is now ready for sewing!
Image from www.tamarembroideries.co.uk.

[7] McNett. (2011). How to thread a needle. *McNett.* Retrieved from www.mcnett.com/gearaid/blog/how-to-thread-a-needle.

TOC

CAUTION: If the knot is too small, it will pull through the fabric and lose our valuable stitches! Practice the knot above, or tie several knots like the one below, on top of one another (Figure 28).

Figure 28: Doubling or tripling this simple knot is also an option
Image from www.styleengineers.org.

3. Conductive Sewing

Woven fabric usually has a natural front and a back, like our clothes have an inside and an outside. View this YouTube video if we're unsure about which is the front or back side of the fabric we are using: http://bit.ly/fabricsides. This should be a strong consideration when we modify wearable material with electronics.[8]

Prepping to sew:

Step 1. Consult the circuit diagram to lay out all of the components. Use a pencil to sketch out directly onto the fabric, or tape the components down. Sew through the tape or remove it later.

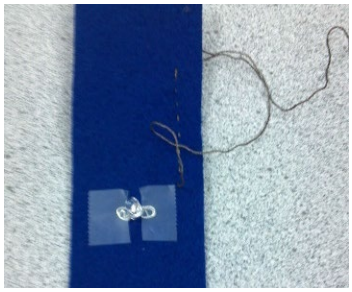


Figure 29: Step 1. Tape components into place before sewing
Image from http://blog.usu.edu/: Craft Technologies 2013.

Step 2. Before sewing, make sure the knotted end (the tail) of the thread is pulled long, and that there's enough length (at least two inches) on the other end of the thread so that the unknotted side doesn't slip out of the needle. The thread below is way too short!

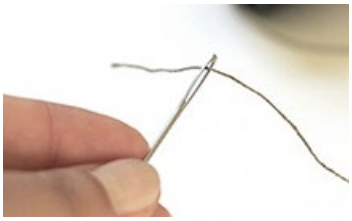


Figure 30: Step 2. This is too short! Pull this end out a little before sewing
Image from www.lilypadarduino.org.

Step 3. Figure out where to make the first stitch, we usually start by sewing one of the components into place. In order for the knot to appear in the backside of the fabric, the needle has to push through the fabric from the back.

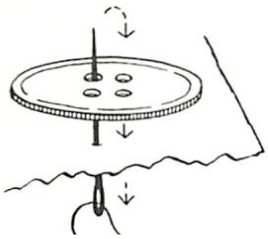


Figure 31: Step 3. Make a stitch without pulling the knot through the fabric
Image from www.styleengineers.org.

Step 4. Pass the needle through the component (in this case, a button). Pull through to make sure the thread is taut, but don't pull so hard that the knot pulls through the fabric or the thread breaks.

[8] Lilypad. (2013). Sewing basics. *Lilypad.* Retrieved from http://lilypadarduino.org/?page_id=1256.

TOC

Step 5. Hold on to the fabric and component, they are not secured yet! Make a small stitch, by gently placing the needlepoint on the other side of the component we're trying to secure and pull the thread through to the backside of the fabric. The needle should emerge right next to the knot.

Step 6. Make sure the thread is taut before each time the needle is passed through the fabric. There shouldn't be any loops in the thread, and we shouldn't be pulling the material so much that the fabric wrinkles or the thread breaks. Check both the front and back of the fabric that there is no excess thread (Figure 32). Untangle any knots, as loopy threads can cause short circuits (Figure 33).

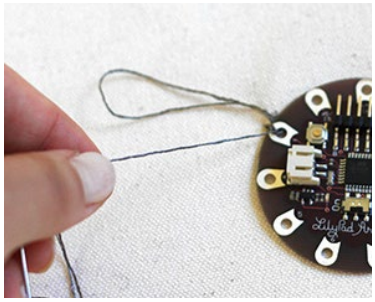


Figure 32: Check both the front and back of the fabric for excess thread
Image from www.lilypadarduino.org.

Figure 33: Don't leave loops of thread like this on either side of the fabric. Pull the thread taut
Image from www.etextilelounge.com.

Step 7. Pass the needle through the same spot three times (make three loops). This is to make sure the conductive thread has a secure electrical connection with the component.

Figure 34: Step 7. Examples of components being sewn through three times.
Images from www.adafruit.com, www.bareconductive.com, & www.sparkfun.com, CC BY-NC-SA 3.0.

Step 8. After we've looped three times, consult the blueprint and figure out where to stitch. This line needs to connect to something, right?

Felt does not have a front or back distinction because of the way it is manufactured. While that may make things easier for beginning crafters, this feature may also become a hindrance, because it's easy to mix up whether we're stitching on the front or the back. When the fronts and backs are mixed up, we may end up sewing circuits to the wrong components or to the wrong polarities from what we intended in the circuit diagram.

Before making the first stitch, decide which will be the front side of the fabric, and make a mark on the backside so we don't forget. LEDs should generally be in the front; we can choose where the battery and other components will go.

REMEMBER: "Positive to positive, negative to negative." A common error is to sew through one end of a component to the other end of the component. That would result in a short circuit.

TOC

4. Running Stitches

Sewing tips:[9]

Step 1. Put tape down to help sew in straight lines.

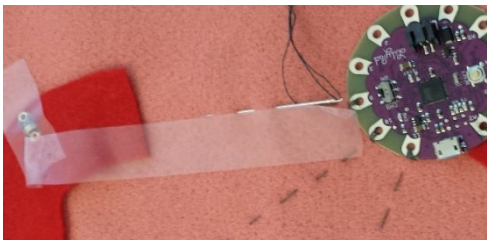


Figure 35: Step 1. Tape can be a guide for your needle.
Image by Tomoko Nakajima

Step 2. Use a running stitch to get the needle and thread to the point that was just marked. That means the needle will pass through in and out of the fabric. Notice how this makes a dotted line. Try to make the stitches even by making sure the needle follows a straight line to that point, and by gathering the same amount of fabric each time. Make small stitches, no larger than a pinkie fingernail, and pass the needle carefully in and out, stab "up and down."

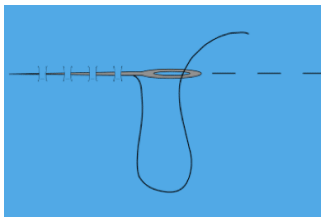


Figure 36: Step 2. Make evenly spaced stitches.
Image from www.styleengineers.com.

Step 3. Learn how to use both hands when stitching! Use one hand for the needle and hold the component and/or the fabric in the other hand.

> **CAUTION:** We may be tempted to make huge stitches to save time. However, larger stitches leave more gaps and loopier thread, which may cause short circuits (see below).

Figure 37: These stitches are a little big…
Image from www.kitronik.co.uk.

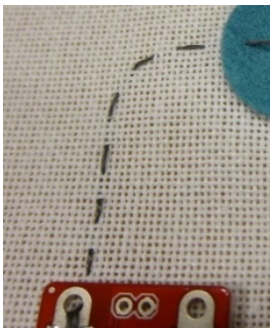


Figure 38: These stitches are small, evenly spaced, and appear secure
Image from www.sparkfun.com, CC BY-NC-SA 3.0.

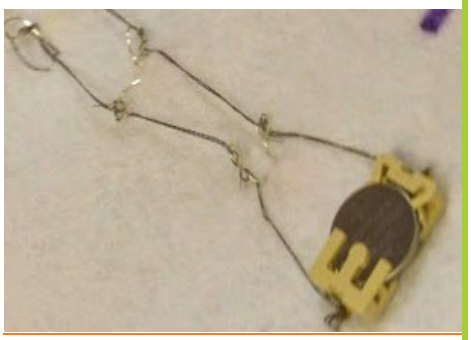


Figure 39: These stitches are too loose and are not well secured to the fabric
Image from www.librarymakerspace.blogspot.com.

[9] DMC. (2014). Embroidery stitch guide. *DMC USA.* Retrieved from www.dmc-usa.com/Education/Technique-Overviews/Embroidery/Embroidery-Stitch-Guide.aspx?technique=embroidery.

TOC

Step 4. While sewing, pull the thread taut on both the front and back of the fabric. There should not be any bubbles or wrinkles in the fabric or the thread. To create smooth uniform stitches, pull each stitch through with the same amount of tension. If the stitch is too loose the stitch will appear limp and if the stitch is pulled too tightly the fabric will pucker and the design will be distorted.

Figure 40: Step 4. Try not to let stitches bubble up like this
Image from www.etextilelounge.com.

Step 5. Prevent the thread from twisting by turning the needle a slight quarter to half turn with each stitch. If the thread gets twisted while stitching, drop the threaded needle and let it hang freely until it "unwinds."

Step 6: To create a secure electrical connection, remember to loop through the fabric and component three times.

Step 7: If this line makes another connection, change direction and sew toward the next point on the blueprint. If not, make an end knot.

NOTE: As we sew, the thread may run out or break. It happens! Cut more thread, wax it, and start again. We can even tie the new thread into a knot with the old thread and keep going. If the thread gets tangled, try to gently undo the knot (don't pull!). If that doesn't work, cut the knotted part off and splice the thread with new thread (Figure 41 & Figure 42).

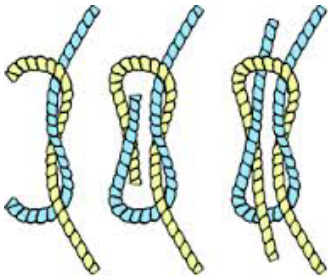


Figure 41: Use this simple knot to tie the old thread with the new thread and make a splice
Image from www.en.wikibooks.org:
Adventist Youth Honors Answer Book

Figure 42: An example of spliced thread
Image from www.ebay.com:
handtools-uk

5. Making end knots

Stitch to the place where the thread should be tied off. Decide whether to have the knot be visible on the front or hidden in the back. Make sure the needle is on the side of the fabric where we want to make the knot.[10] This quick video demonstrates how to tie off the thread: http://bit.ly/tie-off-thread.

[10] Kitronik Ltd. (n.d.). Getting started with e-textiles: Finishing off your stitches. *Kitronik Ltd.* Retrieved from www.kitronik.co.uk/blog/getting-started-e-textiles-finishing-stitches/.

TOC

How to make an end knot:

Step 1: Insert the needle under the last stitches created (Figure 43). We can also make a light stitch through a small amount of the fabric instead.

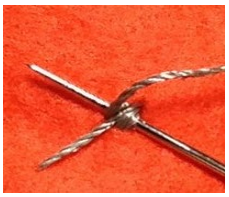


Figure 43: Make one more stitch
Image from www.kitronik.co.uk.

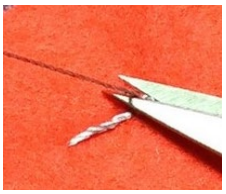


Figure 44: Cut the leftover thread
Image from www.kitronik.co.uk.

Step 2: Repeat that light stitch in the same spot, do it three times, pulling the thread taut each time.

Step 3: Cut the thread off close to the knot.

6. Taking out stitches

Oh no! We made a mistake! It happens to everyone. Before scrapping the entire project, think carefully about the easiest and most effective way to fix the problem. Sometimes just "taking out" or remove some stitches would do the trick. We might use a seam ripper to carefully cut the conductive thread without re-doing the entire circuit (Figure 45). Remove the bits of cut thread completely from the project to prevent shorts.

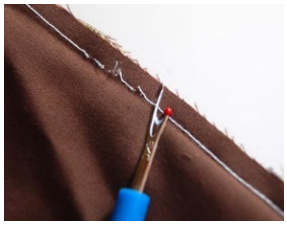


Figure 45: Use a seam ripper to gently cut parts of the sewn thread.
Image from www.sewingloftblog.com.

WARNING: Be careful, the point and the concave dip between the seam ripper points are very sharp!

Alternatively, we could also use thread snips or other detail scissors for taking up stitches (Figure 46).

Figure 46: Thread snips and scissors
Images from www.nordicneedle.com, www.firemountaingems.com, & www.ebay.com/handtools-uk

7. Storing works in progress

When we take breaks from sewing, we gently secure the needle in "stand-by" mode by tucking it lightly into the fabric (Figure 47). Don't ever let the needle hang loose or the needle will get lost!

Figure 47: Tuck the needle into the project before putting it away ortaking a break.
Image from www.makezine.com: Nataliezdrieu.

TOC

8. Finishing touches

Dab some craft glue or nail polish on the knots to secure them (Figure 48).

Figure 48: Using nail polish to secure the knot
Image from www.kitronik.co.uk.

CAUTION: Don't glue over knots until the circuit has been tested and it works. If the circuit is not functional, those stitches might have to come out later.

CAUTION: Too much glue will make the thread no longer conductive!

CAUTION: Check http://bit.ly/gluesafety for potential hazardous ingredients before introducing glues into the classroom.

TOC

Coding

Once we get to the programmable Circuit Playground projects, there are three software options to program the wearable microcomputers we use in e-textiles: ModKit, Arduino, and Codebender. All of these applications are free, and we only need one! [11]

1. Programming software: Modkit

Download Modkit: http://bit.ly/DLmodkit

Modkit Micro™ is a drag & drop programming environment that makes it easy to program electronics. Because it is block-based rather than text-based, this platform is a convenient transition for students accustomed to programming in Scratch or the Lego Mindstorms. Modkit also features *Code View*, which allows the user to also begin to learn what the commands look like in text form. Unfortunately, we have experienced some issues with loading this software, and have decided not to encourage its use in e-textiles classrooms (updated October 2015).[12]

2. Programming software: Arduino

Download Arduino: http://bit.ly/DLarduino

The open-source Arduino Software (IDE) is written in Java and based on Processing and other open-source software. This programming language is widely used by people making interactive projects, because Arduino boards can read *inputs* (a light on a sensor, a finger on a button, or a Twitter message) and turn them into *outputs* (activate a motor, turn on an LED, publish something online). This environment is commonly used in many areas of computer science and is a great introduction for students into vast world of computer programming. Unfortunately, for these reasons, it may be very difficult for people with no programming experience to get used to Arduino. The program may also require users to troubleshoot by searching for coding "bugs" or mistakes that can be frustrating for novices and experts alike.[13]

3. Programming software: Codebender

No need to download! Use the online platform: https://codebender.cc/.

With Codebender, all we need is a browser. It looks just like Arduino (and in fact, it is Arduino). This is a great choice if the computer does not have hardware space for installing & maintaining software. Like the online version of Scratch, all our code is stored on the cloud and we can open and modify our work anywhere with Internet access, as long as each person registers as a user. Use Codebender only

[11] LilyPad. (2013). Setup. *LilyPad.* Retrieved from http://lilypadarduino.org/?page_id=1673.
[12] Modkit. (2016). *Modkit* [website]. Retrieved from www.modkit.com.
[13] Arduino. (2016). *Arduino* [website]. Retrieved from www.arduino.cc.

TOC

if there is fast and reliable online access. Install the plugin so that Codebender can communicate with the USB ports.[14]

4. Setting up the Circuit Playground for programming

Regardless of software ultimately chosen, we will need to set up the system to communicate with the Circuit Playground microcomputer, developed by Adafruit. In this unit we use the Circuit Playground Classic. Other sewable microcontrollers are available but generally cost more or require more parts. For instance, you may also use a Sparkfun LilyPad (does not include switches or sensors onboard – you will need to order those separately) or the Adaruit Circuit Playground Express (costs $5 more but also works with Make:Code programming environment).

Circuit Playground

You will need a micro-USB cable (not included in the kit; make sure it has not just power transfer but data transfer capabilities). For more information on the Circuit Playground see: http://www.adafruit.com/product/3000 and http://learn.adafruit.com/introducing-circuit-playground.

Connecting to the computer:

Step 1. Open the programming platform (ModKit, Arduino, or Codebender).

Step 2. Select the correct serial port so the software can talk to the LilyPad.

For Mac OS
In Arduino, under the "Tools–>Serial Port" menu, find the entry that looks *something* like this: "/dev/tty.usbserial-A900J2Q7." Select this port.

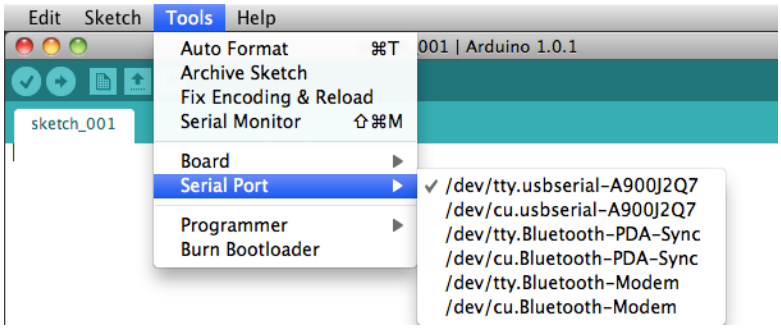


Figure 49. Image from http://lilypadarduino.org/.

For PC
In Arduino, under the "Tools–>Serial Port" menu, choose the highest numbered COM port (it is *never* Com 1).

[14] Codebender. (2016). *Codebender* [website]. Retrieved from www.codebender.cc.

TOC

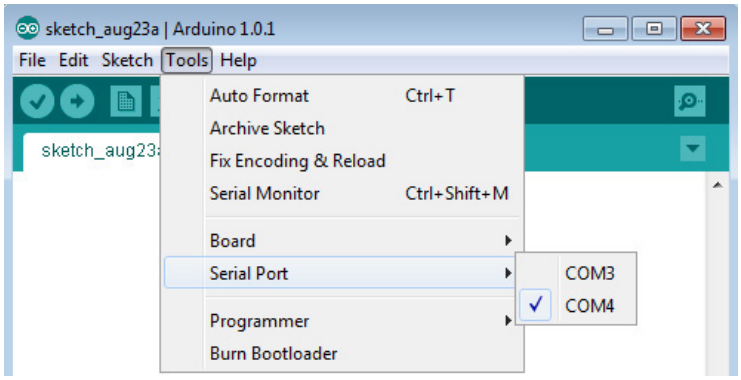


Figure 50. Image from http://lilypadarduino.org/.

Step 3. Select the correct board so that the software knows we're using an "Adafruit Circuit Playground" and not a different Arduino board. Under the "Tools–>Boards" menu, select the entry appropriate for the board we're using:
Adafruit Circuit Playground

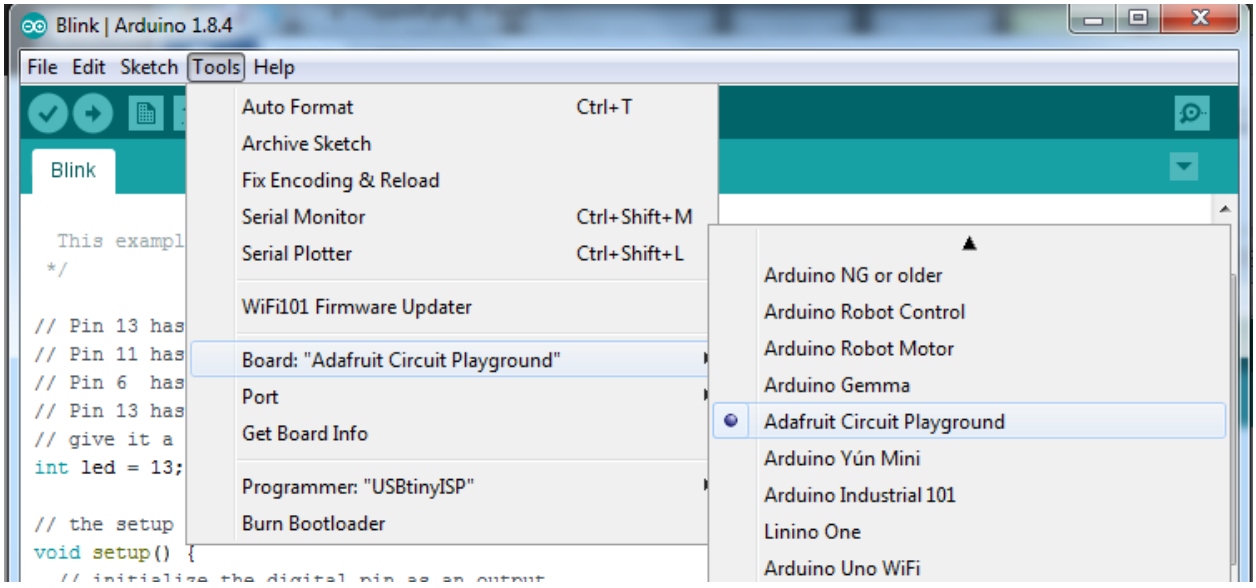


Figure 51. Image from https://learn.adafruit.com/introducing-circuit-playground/set-up-test-arduino.

NOTE: You may have to select and re-select the port and board on occasion, especially when you plug in a device.

TOC

TROUBLESHOOTING

1. Connecting the Circuit Playground to the Computer

Does the computer still have trouble communicating with the Circuit Playground? If we tried all the steps above and the correct port does not appear, start troubleshooting by following the suggestions below in order.

Start-up sequence issue: On some computers, the Circuit Playground has to be plugged in *before* opening the programming software. Close the programming software completely, unplug and plug the Circuit Playground back in, then start the software again. Check if the port selection changed at all.

Computer port issue: There might be a problem with the USB port we are connecting to, this seems to happen more often with PCs. This apparently happens because the operating system (OS) automatically "finds" and installs drivers for Arduino boards, but they almost always install the wrong ones! Close the software, then plug the Circuit Playground into a different USB port. Open the programming software again and see if a new port selection is available. Also, be sure that the cable connection is secure on the Circuit Playground and also in the computer port.

Port identity issue: When using the USB ports on the computer for other items (wireless mouse, keyboard, etc.), there may be more than one option to select from in the port pull-down menu. Unplug the board, observe all the ports that are there, then plug the Circuit Playground back in to see which new port appears in the menu. Select that one.

USB cable issue: If the serial port is still not appearing in the menu, try a different USB cable. There may be slight differences in capacity between a USB cable that's used for cellphone charging and one that's used for development purposes. If we try a different cable and it works, be sure to set aside the original one so that it doesn't get used again for e-textiles. This happens often enough that it is estimated that this is the source of the problem about 50% of the time.

System overload: Sometimes the computer is just overwhelmed! Save all work and restart the computer. Be sure to plug the Circuit Playground into the computer first before opening the programming software.

But it was working before!: If we were able to program the board before, but it's not connecting now, check the port and board selection again. Try selecting and re-selecting the port and board every time new code is uploaded to the Circuit Playground. We hope Arduino will have this bug fixed soon!

Material in the Troubleshooting section was adapted from Sew Electric and customized for use with the ECS curriculum.

TOC

Bad Circuit Playground: When plugging the board in to the computer, a light should turn on and blink. On the Circuit Playground, this light is marked STAT. Double-check that the board works by programming just the small LED on pin 13 (try the "Blink" code under Arduino Examples under the file menu). Every so often, we have a Circuit Playground that doesn't work properly because it was shorted in the past or has some other bug that we cannot fix. Static electricity might also affect these microcomputers. Try the code with a different Circuit Playground by using alligator clips to test the program on the e-textile. We may have to cut the original Circuit Playground out and sew in a different one that works.

2. The four stages of programming

We will focus here on the text-based programming platform. If using ModKit, knowledge from Scratch will help develop programs easily in that environment. Programming the Circuit Playground to do what we want takes place in four stages in Codebender/Arduino:

1. Write a program
2. Compile/Verify the program
3. Load the program onto the Circuit Playground
4. The program executes on the Circuit Playground [15]

First, **write** the program. In the Codebender/Arduino environment, we code in a programming language that is a variant of the C/C++ languages. Next, **compile** or verify the program. In Arduino, hover the mouse over the Verify/Compile icon (the check button in upper left corner), the word "Verify" appears at the top of the window. In Codebender, click the "Verify" button. This simultaneously does two things. First, it translates the code that we wrote to code that the computer chip on the Circuit Playground can understand. Also, if we've made any mistakes in the code, they will be detected during this phase and we have to "de-bug" them (eliminate all of the programming errors) and recompile the code before we move on to the next step.

The compiling process takes a few seconds. Once it's done, Arduino says it's "Done compiling" and tells us the size of the sketch (the amount of memory it takes up). Then we **load** the compiled code onto the Circuit Playground, officially moving the code from the computer onto the microcontroller. The Circuit Playground should then **execute** (run) whatever we coded it to do. If the Circuit Playground and the components in the circuit are not behaving the way we expected, we have to de-bug those errors, too. The program will only do what we told it to do, so if something is happening that is unexpected, think through exactly how we expect that code to run and compare it to our program code.

[15] Adapted from LilyPad. (2013). Basic programming steps. *LilyPad.* Retrieved from http://lilypadarduino.org/?page_id=209.

TOC

3. Arduino program structure

Each Arduino program has three main parts: a section where we declare variables, a "setup" section and a "loop" section. When the program executes, it first **defines** (names) the variables, then executes the **setup section** (initialization section) once, and then executes the **loop section** (activity section) over and over. (There may also be a "building blocks" or **function definition** section after that where special procedures can be added. We discuss this more in the Human Sensor Project).

Here is an example program in the Arduino window (this looks very similar in Codebender):

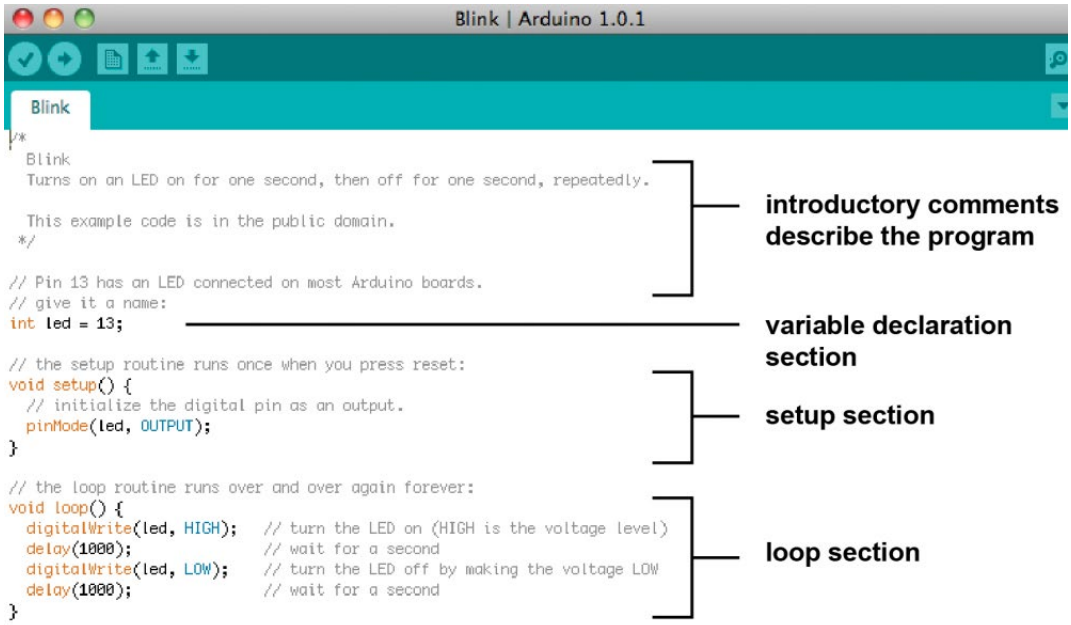


Figure 52. Image from http://lilypadarduino.org/.

4. Programming basic commands

All programs execute in a sequential fashion, obeying the written commands (code) line by line in order as they are written from top to bottom. There are a few different types of commands or code elements in most programming languages. The most common are **comments**, **simple statements** and **conditional statements**. Here we describe what these elements look like in Arduino. Go to Arduino Language Reference (https://www.arduino.cc/reference) for more detailed information.

Comments are pieces of code that are ignored by computers. Use them to make notes in the code, but they don't affect the behavior of the program. Comments are useful for reminding us what the code is supposed to do in the program that we wrote two months ago. They can also communicate things to people who will see the code in the future. Anything written on a line after two slash characters // is a comment. Comments show up a greyish brown in the Arduino window. Here are a couple example comments:

TOC

```
// This is a comment. It doesn't do anything.
// Comments just make code easier for us and other people to understand
```

Anything written between "/*" and "*/" characters is also a comment. Comments that start with "//" can only be on one line, but comments between the "/*" and "*/" characters can take up several lines. Comments always appear in a greyish brown color in Arduino.

```
/* This is a comment. It doesn't do anything.
Comments just make code easier for us and other people to understand */
```

A **simple statement** is an instruction. It takes up one line and is *always followed by a semi-colon*. The semi-colon tells the computer that it is done with that line, now go to the next line. On the left below are some examples, followed by comments on the right that describes what the statement does.

```
int LEDPin = 13;                  // Declares a variable called "LEDPin" and connects pin 13 to this variable.
int switchPin = 2;                // Creates a variable called "switchPin"and connects pin 2 to this variable.
int switchValue;                  // Creates a variable called "switchValue", but does not initialize it
delay(1000);                      // Tells the LilyPad to do nothing for 1000 milliseconds (1 second).
digitalWrite(LEDPin, HIGH);       // Sets LEDPin (pin 13) to +5 volts or HIGH to turn the LED on.
digitalWrite(LEDPin, LOW);        // Sets LEDPin to 0 volts or LOW to turn the LED on the LilyPad off.
```

For a more thorough explanation of what each simple statement does, and to find more commands, go to the Arduino Language Reference.

Conditional statements are statements that consist of a condition and then a series of statements in braces like this { } that execute only when that condition is met. Here is an example of an **if conditional**. Notice how it consists of the word "if" and then a condition in parentheses ( ). We would read this: "if the variable switchValue is equal to LOW, then turn the LED on." The two equal signs tells the microcomputer to *check* what the value is. One equal sign *sets* the value to what follows after.

```
if (switchValue == LOW) {         // This line is the condition, followed by the opening brace
    digitalWrite(ledPin, HIGH);   // This tells an LED to turn on
}                                 // Finally, there is a closing brace
```

The example below is a **while conditional**. This statement reads: "While the variable switchValue is equal to LOW: Turn the LED on, then do nothing for 100 milliseconds (1/10th of a second), then turn the LED off, then do nothing for 100 milliseconds, then store a new value into the variable switchValue."

TOC

```
while (switchValue == LOW) {                // This line is the condition, followed by the opening brace

    digitalWrite(ledPin, HIGH);             // This tells the LED to turn on
    delay(100);                             // This tells the LilyPad to wait for 100 milliseconds
    digitalWrite(ledPin, LOW);              // This tells the LED to turn off
    delay(100);                             // Wait another 100 milliseconds
    switchValue = digitalRead(switchPin);   // This stores a new value into the variable switchValue
}                                           // Closing brace
```

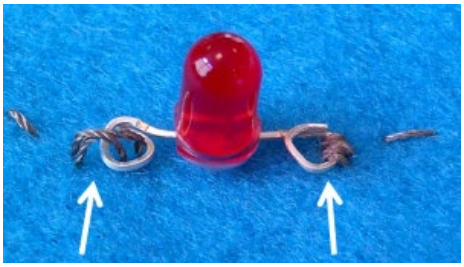

TOC

Electrical Problems

Electrical problems arise from either the design or construction of the electrical circuits. These are physical problems that we'll need to fix by hand with scissors, a needle, thread, fabric, or glue. The next section provides examples of common problems in the circuitry, information about how to find them, and step-by-step instructions for fixing them. [16,17]

There are different types of problems that affect the circuitry:

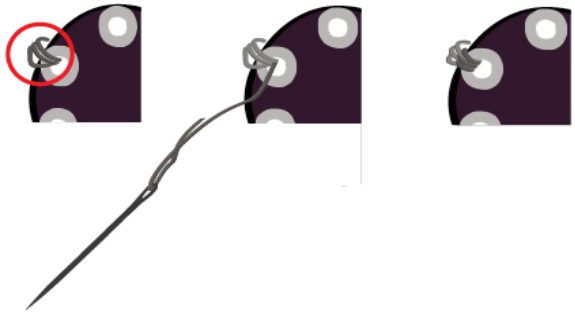
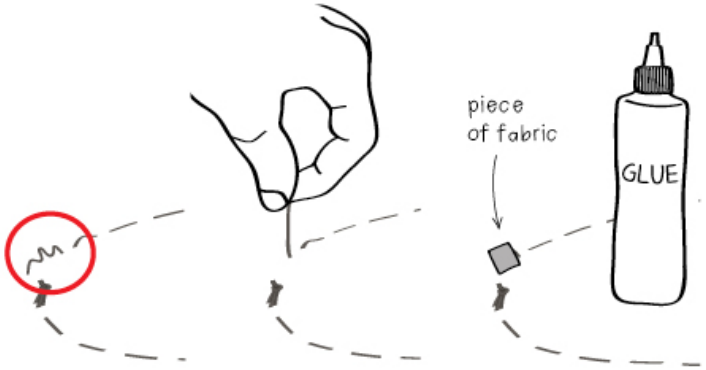
- Loose connections
- Short circuits
- Reversed polarity
- Broken components

SYMPTOM	LOOSE CONNECTIONS
Parts of the project flicker or only work some of the time: **Loose Connection**	**Loose Connections** occur when the thread that is stitched through a component (like an LED, speaker, Circuit Playground, battery holder, etc.) is too loose. See Figure 53 for a visual example. If the thread is loose, there will not be a consistent electrical connection between the thread and the component. To carry electricity through the circuit, the thread must be tightly pressed up against the metal pins of the component. Loose connections can be caused by loose stitching or unraveling knots. If a knot is *not* secured with glue, it can come undone, loosening the connections near it.  *Figure 53: Loose connection on the left, secure connection on the right* Image from www.kitronik.co.uk. **Check for Loose Connections:** Gently bend and stretch the project. If this causes the LED to turn on and off, a loose connection is likely. Look carefully at the connections between the thread and the components, and check both the front and back sides of the fabric. Make sure each connection is snug and tight. If there is loose thread around any metal pin or an unraveling knot, we'll need to fix it. **Fixing Loose Connection Points:** To fix loose connections at component pins, thread the needle with conductive thread. From the back or underside of the fabric, push the needle up through the pin with the loose connection. Loop through the pin a few times. Make sure

[16] Buechley, L., Qiu, K., Goldfein, J., & de Boer, S. (2013). *Sew electric: A collection of DIY projects that combine fabric, electronics, and sewing* [online version]. Retrieved from www.sewelectric.org/troubleshooting/electrical-problems/.

[17] Kitronik (n.d.). Finding fault in e-textiles. *Kitronik Ltd.* Retrieved from www.kitronik.co.uk/blog/fault-finding-in-e-textiles/.

TOC

SYMPTOM	LOOSE CONNECTIONS
	the thread is touching the original stitches we sewed in several places. Push the needle to the back or underside of the fabric. Tie a snug knot, making sure that the new thread is pulled tightly against the pin and the old stitching, and secure the knot with a little bit of glue. Remember not to add too much glue, since that can also cause connectivity issues.  *Figure 54: Adding tighter loops to fix a loose connection* Image from www.sewelectric.org, CC BY-NC-SA 3.0. **Fixing Unraveling Knots:** If there is a knot that is unravelling, find the end of the thread and pull it to re-tighten. Cut out a small piece of fabric and glue it down over the unravelling thread to hold the thread in place. Some stitches may have to be resewn if they have come undone, and they were meant to be connected. Tie the unraveled thread to the new thread in several places to make a solid electrical connection. Glue (in moderation) may also make those connections more secure (Figure 55).  *Figure 55: Securing down thread that unraveled* Image from www.sewelectric.org, CC BY-NC-SA 3.0.

SYMPTOM	SHORT CIRCUITS
Parts of the project don't work, only work some of the time, or flicker on and off: **Short Circuit**	**Short Circuits** or "shorts" happen when two threads that should not touch come into contact with one another, creating a convenient "shortcut" for the current. Electricity will always travel the path of least resistance, so if there is a shortcut, it will bypass the lights, sensors, switches, etc. So the power (+) and ground (-) traces in a circuit should never ever touch one another in e-textiles. It is okay for traces of the same polarity to be touching each other, though. For instance, all of the threads attached to the ground (-) pin can touch each other, but the positive (-) thread attached to Pin 9 should never touch the thread attached to the ground (-) tab. This can get very confusing in the more advanced projects where we might program some of the Circuit Playground pins to be negative and the others to remain positive. Always remember: negative to negative, positive to positive!

TOC

SYMPTOM	SHORT CIRCUITS

Removing Loose Threads: From the back of the fabric (if possible), pull on the loose thread until it is gathered in one spot. Cut out a small piece of fabric, apply a small amount of glue to it, and tuck the extra thread behind it, making sure that the extra thread no longer touches any neighboring traces or pins (Figure 56). If there is a lot of excess thread, we may want to tie it into a tighter knot instead and cut off the excess.

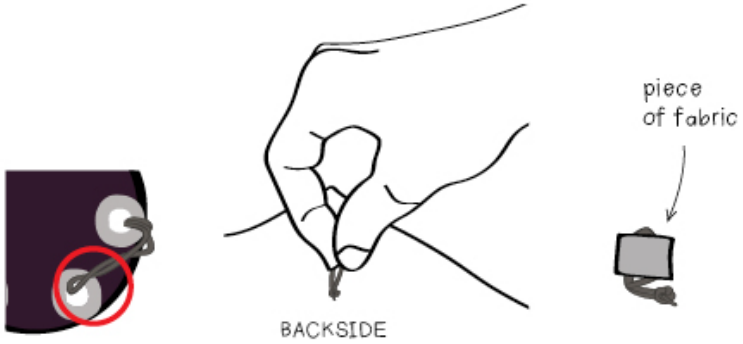


Figure 56: Moving excess thread away from other traces or pins
Image from www.sewelectric.org, CC BY-NC-SA 3.0.

Eliminating Long Thread Ends: To eliminate long knot tails, trim them down to 1/4" (6mm) or shorter. Once we know that those traces function correctly, seal the knots with glue so that they do not come unraveled. Make sure that the tails do not brush up against any neighboring traces. We can also glue a small piece of fabric down over the knot tails to keep them from unravelling and touching other traces (Figure 57).

Figure 57: Eliminating thread tails that are too long
Image from www.sewelectric.org, CC BY-NC-SA 3.0.

Fixing a Stitched-Across Component: If we have sewn the (+) side of a component to its (-) side, we will need to cut the thread that is connecting the two pins. We will be left with two very short threads. Tug on each thread tightly, move it away from the other thread, and glue it down with a small piece of fabric (Figure 58).

TOC

SYMPTOM	SHORT CIRCUITS

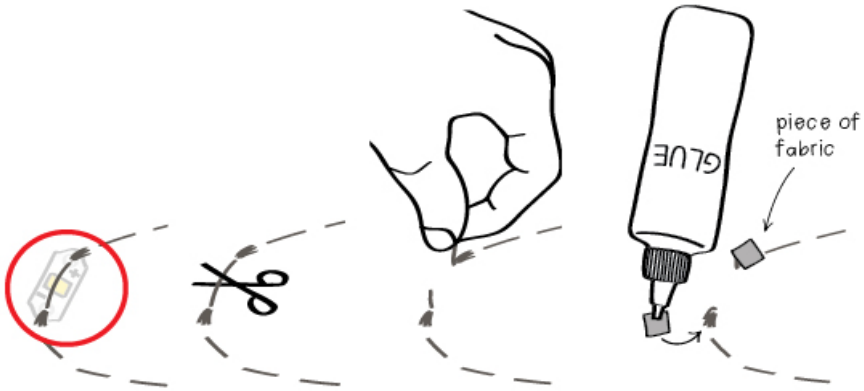


Figure 58: Separating the thread connecting the (+) and (-)
Image from www.sewelectric.org, CC BY-NC-SA 3.0.

Troubleshooting Parallel Circuitry: Another common "stitch across" error happens for novice crafters when transitioning from a single LED to building a parallel circuit. It's natural to want to do all the sewing at once, to try to connect everything without cutting the thread and tying unnecessary knots. In Figure 59, the circuit functions perfectly with one light, but in extending out to add another light (dotted lines), the red (positive) and the black (negative) traces were connected without leaving space for the component. This will result in a short, and none of the circuit will work. To fix this, create a small gap for the second LED by cutting the thread and tying the thread ends onto the new LED wires. Reinforce the new connections with extra stitches.

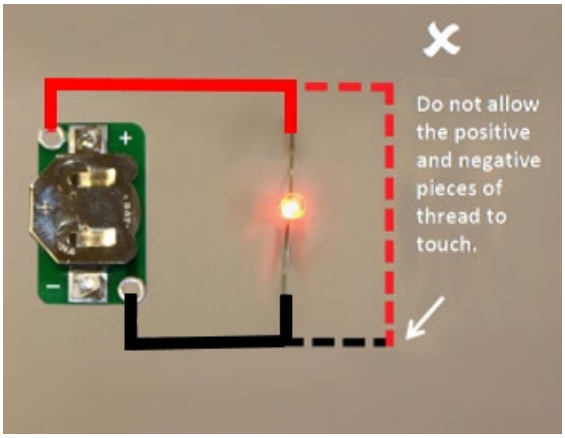


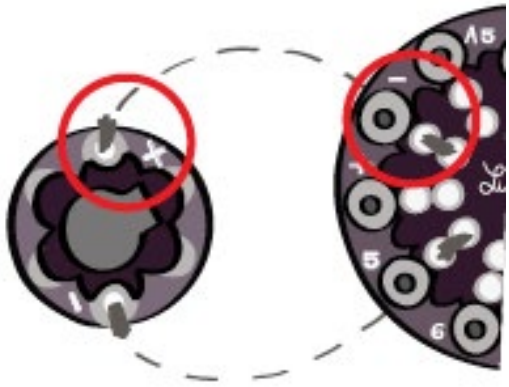
Figure 59: Mind the gap! Remember to leave a "gap" for any components we want to add
Image from www.kitronik.co.uk.

Removing Stitching Across a Battery Holder: People often stitch across the battery holder and cause shorts. How? The piece of metal across the top of the battery holder is positively (+) charged. If we sewed across, we will have to remove the negative trace and stitch around the battery holder. If we did this with the positive trace, the circuit may function, but this part of the circuit should eventually be rerouted because the thread is loose and will eventually catch on something, break, or cause a short circuit.

Fixing Overlapping Stitches: If there is an area in the project where two traces cross and touch, separate the two traces by putting something in between. Glue a piece of fabric or place tape (something non-conductive) in between the two lines to keep them apart.

TOC

SYMPTOM	SHORT CIRCUITS
	 *Figure 60: Fixing two traces that were accidentally crossed* Image from www.sewelectric.org, CC BY-NC-SA 3.0.

SYMPTOM	REVERSED POLARITY
The project does not turn on at all, or areas connected to certain components do not work: **Reversed Polarity**	When we stitch the (+) side of a component to the (-) side of a battery (or the (-) pin of the ProtoSnap), and the (-) side of a component to the (+) side of a battery (or one of the numbered pins on the ProtoSnap), there is **Reversed Polarity** and it will not work. Electricity will only flow through most components in one direction—from the (+) to the (-) side of a component. This mistake can easily occur when stitching on both the front and back sides of the fabric, because the back side of the fabric should be the mirror image (opposite) of the circuit diagram. That means the circuit diagram may show stitching to the right side of the component, but if the fabric is flipped to the back side, that stitching will appear to be on the left side. If an LED is connected backwards, the circuit will not be complete either. If using wired LEDs, remember that the longer wire on the LED is the positive (+) side, and the shorter wire and the flattened plastic mark the negative (-) side. **To Find Reversed Polarity:** Look carefully at each component in the project and compare the front side of the fabric to the circuit diagram. Make sure that each component's negative (-) pin is stitched to the matching negative (-) circuit, and that each component's positive (+) pin is sewn to only other positive (+) components. A common problem is that the battery is inserted upside down (the battery holder is marked as (+) on the top and (-) on the bottom).  *Figure 61: Identifying a reversed component* Image from www.sewelectric.org, CC BY-NC-SA 3.0.

TOC

SYMPTOM	REVERSED POLARITY

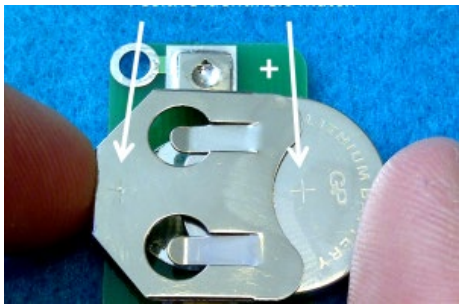


Figure 62: The battery must be inserted with the (+) side up.
Image from www.sewelectric.org, CC BY-NC-SA 3.0.

Fixing Reversed Polarity: Unfortunately, there may not be an easy way to fix reversed polarity. One solution is to remove the component and rotate it into the correct orientation, or undo some of the stitching and reattach the components with some new stitches. If stitches need to be cut to remove the component, think about where to cut to save as many stitches as possible (Figure 63). Maybe new thread can be spliced to the existing thread to minimize extra sewing. Once the component is removed, tape it down in the correct orientation. Double check that it is correctly aligned, then resew.

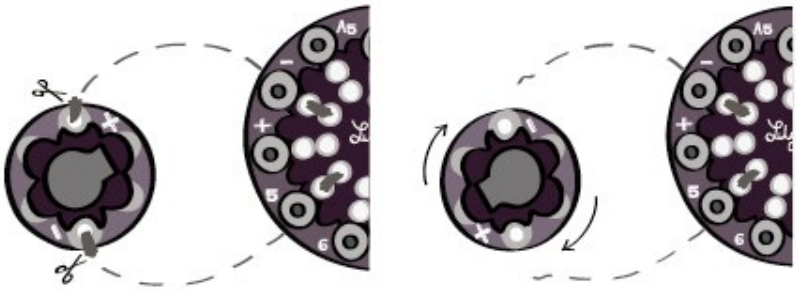


Figure 63: Cutting loose a component that was sewn in backwards
Image from www.sewelectric.org, CC BY-NC-SA 3.0.

Thread the needle with conductive thread. Begin with the (-) trace. On the back side the fabric (if possible), tie the new thread to the old stitching on the (-) trace so the new knot is not visible from the front. Sew toward the reattached component. Make sure the new thread is touching the original stitches in several places. Stitch through the (-) pin of the component three times. Tie a knot on the underside of the fabric, trim its tails, and seal it with a dab of glue. Repeat the same process for the (+) pin of the component.

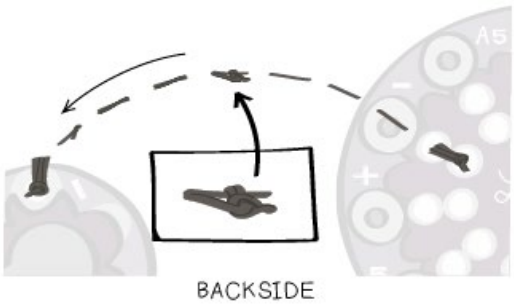


Figure 64: Splicing the cut thread with new thread to reattach a component
Image from www.sewelectric.org, CC BY-NC-SA 3.0.

TOC

Code Problems – Compile Errors

There are two common types of error messages in programming. **Compile errors** are errors that happen when we attempt to compile code and the compiler (i.e., Arduino) finds an error in it. The following examples are all errors from Arduino. The error message box has a scroll bar on the right, and the errors may be longer than expected. Scroll to the top of the black error box at the bottom of the Arduino window. If the name of the program followed by .ino in orange text is at the top of the box, it is a compile error (in this case, Blink.ino is our program).[18]

```
A FUNCTION-DEFINITION IS NOT ALLOWED HERE BEFORE '{' TOKEN
Blink.ino: In function 'void setup()':
Blink:19: a function-definition is not allowed here before '{' token
Blink:24: Correction: expected '}' at end of input
```

Compile errors can be confusing, since the error that it is referencing might not be on the line it says it is. For example, in this error the compiler found a code problem on line 19 and line 24 (Blink:19, Blink: 24). This error could be because the program has the curly brace in the wrong spot, or maybe it was forgotten completely. In this case, it looks like both curly braces were forgotten, since line 24 says that the compiler "expected '}' (a closing curly brace) at the end of input (program). Compile errors often occur because of syntactical mistakes like this. They can occur if the program tries to use a variable that has not been initialized yet, or if there is extra text anywhere in the program, such as a comment line that was accidently un-commented. With a compile error, carefully read the messages in the error box of the Arduino window. Investigate the code where Arduino highlights or jumps to. If there are no visible problems in that area, carefully read through the entire program line by line, looking for the problems described in this section.

Common compile errors

- Missing semicolons **;**
- Missing curly braces **{ }**
- Missing parentheses **( )**
- Missing commas **,**
- Misspellings and mis-capitalizations **DigitalWrite vs digitalwrite**
- Missing variable initializations **int LED; vs int LED=;**
- Extra text in program **/comment** vs **//**comment

[18] Buechley, L., Qiu, K., Goldfein, J., & de Boer, S. (2013). *Sew electric: A collection of DIY projects that combine fabric, electronics, and sewing* [online version]. Retrieved from http://sewelectric.org/troubleshooting/code-problems-compile-errors/.

TOC

Missing semicolons ;

Symptoms

The error says**: Correction: expected ';' before...**. So either in the orange or the black error box, we're missing a semicolon. Arduino usually highlights the line immediately after the missing semicolon to indicate the location of the error.

```
int led = 13

void setup() {
    pinMode(led, OUTPUT);
}
EXPECTED UNQUALIFIED-ID BEFORE NUMERIC CONSTANT
Blink:11: Correction: expected unqualified-id before numeric constant
Blink:13: Correction: expected ',' or ';' before 'void'
```

Here, line 13 states that the compiler expected to find a semi-colon before it started reading the setup content (void setup). It can be fixed by adding the semi-colon after the number 13 on the line above.

```
void loop() {
    digitalWrite(led, HIGH);
    delay(1000);
    digitalWrite(led, LOW);
    delay(1000)
}
EXPECTED ',' OR ';' BEFORE '}' TOKEN
Blink.ino: In function 'void loop()':
Blink:24: Correction: expected `;' before '}' token
```

This example is states that in the void loop function, the last line before the closing curly brace (line 24) is missing its semi-colon. Fix it by adding the semi-colon like so: delay(1000);

Fixing semicolon issues

Look for instances of **Correction: expected ';'**. Replace the missing semicolon and recompile.

Missing curly braces { }

Symptoms

There is a '{' or '}' in the orange or black feedback area. Arduino will sometimes highlight the line immediately after the missing curly brace to indicate the location of the error. Sometimes, unfortunately, it will highlight a completely unrelated line! Grrr...

```
void setup() {
    pinMode(led, OUTPUT);

A FUNCTION-DEFINITION IS NOT ALLOWED HERE BEFORE '{' TOKEN
Blink.ino: In function 'void setup()':
Blink:19: a function-definition is not allowed here before '{' token
Blink:24: Correction: expected '}' at end of input
```


TOC

We can see that the closing curly brace **}** is missing. Add it to fix this bug.

```
void setup()
    pinMode(led, OUTPUT);
}
EXPECTED INITIALIZER BEFORE 'PINMODE'
Blink:16: Correction: expected initializer before 'pinMode'
Blink:17: Correction: expected declaration before '}' token
```

We can see that the opening curly brace **{** is missing, add it to fix this bug.

Fixing curly brace issues

Look for mentions of **'{'** or **'}'** in the error box and correct them. Recompile.

Missing parentheses ( )

Symptoms

The error says **'('** or **')' token** in either the orange or black error box, and may also highlight the line with the missing parenthesis to indicate the location of the error.

```
void setup( {
    pinMode(led, OUTPUT);
}
VARIABLE OR FIELD 'SETUP' DECLARED VOID
Blink:14: Correction: variable or field 'setup' declared void
Blink:14: Correction: expected primary-expression before '{' token
```

This one can be confusing, since the error says nothing about a missing parenthesis. The compiler is calling the parenthesis a "primary-expression". It is saying that if we are not adding any arguments inside of the parenthesis for the function "setup" to use, we need to remember to close the parenthesis to indicate that what is returned is nothing, a.k.a. "void".

```
void loop() {
    digitalWrite(led, HIGH);
    delay(1000;
    digitalWrite(led, LOW);
    delay(1000);
}
EXPECTED ')' BEFORE ';' TOKEN
Blink.ino: In function 'void loop()':
Blink:22: Correction: expected `)' before ';' token
```

Oops! Looks like we forgot a closing parenthesis on line 22.

Fixing missing parentheses ( )

Look for mentions of **'('** or **')'** in the errors. Look for errors that are similar to the ones above. Replace the missing parenthesis and recompile.

TOC

Misspellings and mis-capitalizations

Symptoms

Error will say **'_____' was not declared in this scope**. Arduino may also highlight the line to indicate the location of the error. Also, if one of the built-in Arduino procedures or variables is incorrect, the color of the misspelled or mis-capitalized word will not change to the correct color, **orange** or **blue**; instead it stays **black**. This gives another valuable clue about the cause of the error.

```
void loop() {
    digitalWrite(led, HIGH);
    delay(1000);
    digitalWrite(led, LOW);
    dellay(1000);
}
'DELLAY' WAS NOT DECLARED IN THIS SCOPE
Blink.cpp: In function 'void loop():
Blink:21: Correction: 'dellay' was not declared in this scope
```

When spelled correctly, delay turns orange, indicating that Arduino recognizes this function. Here it is spelled incorrectly and the text stays black. Remove the extra 'L' to fix the problem.

```
void loop() {
    digitalWrite(led, HIGH);
    delay(1000);
    digitalWrite(led, low);
    delay(1000);
}
'LOW' WAS NOT DECLARED IN THIS SCOPE
Blink.ino: In function 'void loop()':
Blink:23: Correction: 'low' was not declared in this scope
```

Again, we can see that 'low' is not capitalized, so Arduino never recognized it or changed the text to blue. Capitalize all of '**LOW**' to fix this bug.

```
if (sensorValue < 1000)
{
    LightPattern1(2000);
}
'LIGHTPATTERN1' WAS NOT DECLARED IN THIS SCOPE
patrick.ino: In function 'void loop()':
patrick:29: Correction: 'Song' was not declared in this scope
```

Here we called LightPattern1 for 2 seconds (2000 milliseconds). It didn't work because our variable is named **lightPattern1** not **LightPattern1**. If that is the case, correct the case.

Fixing misspellings and mis-capitalizations

Correct the misspelling or mis-capitalization and recompile.

TOC

Missing variable declarations

Symptoms

Errors say **'_____' was not declared in this scope**. Arduino highlights the first line in our program that uses the missing variable.

```
// missing int led=13; line

void setup()
    pinMode(led, OUTPUT);
}
'LED' WAS NOT DECLARED IN THIS SCOPE
Blink.ino: In function 'void setup()':
Blink:4: Correction: 'led' was not declared in this scope
Blink.ino: In function 'void loop()':
Blink:9: Correction: 'led' was not declared in this scope
```

This program shows two errors, one in the setup function and one in the loop function. Both of them are caused because we forgot to declare the variable led and link it to a specific pin on our board. When the program tries to use it, it can't find it. We can fix this by adding **int led = 13;**

```
int led1 = A4;
int led2 = 5;
// missing int aluminumFoil=A2; line
int sensorValue;
'ALUMINUMFOIL' WAS NOT DECLARED IN THIS SCOPE
patrick.ino: In function 'void setup()':
patrick:18: Correction: 'aluminumFoil' was not declared in this scope
patrick.ino: In function 'void loop()':
patrick:24: Correction: 'aluminumFoil' was not declared in this scope
```

The commented-out line that is highlighted was done to show what the error looks like if we forgot to declare the variable **aluminumFoil**. We also initialize it to pin **A2** which is one of our analog pins on the board.

Fixing missing variable definitions

Check for missing or misspelled variable definitions at the top of the code. Correct or add the necessary definitions and recompile.

Extra text in program

Symptoms

A common extra text error is a mistyped comment—a comment that is missing a '/', '/*' or '*/'. If there is a mistyped comment, its color will be black instead of grey, giving us a clue to the problem. Arduino may highlight the line where it has found extra text.

TOC

```
void loop() {
    digitalWrite(led, HIGH); x
    delay(1000);
    digitalWrite(led, LOW);
    delay(1000);
}
'X' WAS NOT DECLARED IN THIS SCOPE
Blink.ino: In function 'void loop()':
Blink:9: Correction: 'x' was not declared in this scope
Blink:10: Correction: expected `;' before 'delay'
```

```
void setup() {
    / this is Jaime's pattern
    digitalWrite(led, HIGH);
    delay(1000);
    digitalWrite(led, LOW);
    delay(1000);
    {
    ...
EXPECTED PRIMARY-EXPRESSION BEFORE '/' TOKEN
banner.ino: In function 'void setup()':
banner:9: Correction: expected primary-expression before '/' token
```

Here we forgot the second **/**, which would turn the text grey and cause our comment to be ignored by the computer.

Fixing extra text problems

Correct the typos or remove the extra text from the code. Recompile.

TOC

Code Problems – Logical Errors

When the code compiles and uploads, but it doesn't behave the way that we want it to, we might have a *logical error*. These errors are tricky to troubleshoot because Arduino doesn't give us any feedback about what might be causing the problem, like it does with compile and upload errors. The code compiles and the Circuit Playground is following the code exactly as we have written it, but there is an error in the code that is making the project behave in an unexpected way. If the project doesn't do anything or keeps doing the wrong thing, there might be a logical error.

Finding logical errors in code

First go through the Troubleshooting: Electrical Problems section to eliminate the possibility of circuitry issues. Then check the code for logical errors. The Circuit Playground does exactly what the program tells it to do, so identifying if there is a mismatch between what we want it to do and what the program is telling the board to do can be helpful. Having another person "read" the code and tell us what it is supposed to do and in what order can help diagnose this bug. The errors listed below are the most common; however, there is an infinite variety of logical errors and this guide cannot cover them all.

- Missing variable initializations
- Variables that do not match circuit
- Conditions that are always true or false
- Extra or misplaced semicolons
- Incorrect variable initializations
- Problems with delay statements

Missing variable initializations

Symptoms

A component either does not work at all, or barely works. If there is a very dim LED making very faint sounds, there might be missing a **pinMode(component variable, OUTPUT);** statement. This can also be an INPUT.

EXAMPLE	CORRECTION

`int led = 13;`
 	
`void setup() {`
`}`
	`int led = 13;`
	
	`void setup() {`
	`  pinMode(led, OUTPUT);`
	`}`

TOC

If a sensor or switch gives random or erratic readings, there might be missing a **digitalWrite(component variable, HIGH);** statement. This can also be **LOW**.

EXAMPLE	CORRECTION

`void setup() {`
`  pinMode(led, OUTPUT);`
`  pinMode(speaker, OUTPUT);`
`  pinMode(aluminumFoil, INPUT);`
`  Serial.begin(9600);`
`}`
	`void setup() {`
	`  pinMode(led, OUTPUT);`
	`  pinMode(speaker, OUTPUT);`
	`  pinMode(aluminumFoil, INPUT);`
	`  digitalWrite(aluminumFoil, HIGH);`
	`  Serial.begin(9600);`
	`}`

Fixing missing variable initializations

Look in the **setup** function. Each component must be initialized in setup with a **pinMode** statement. Each sensor or switch should also have a **digitalWrite(component variable, HIGH);** statement in setup. It could also be set to **LOW** depending on what the program needs to do. Add the missing statements and recompile.

Variables that do not match circuit

Symptoms

A component in the project is not working at all because the code does not match the circuit. For example, if the LED is connected to pin 6, but the code is written as though the LED is sewn to pin 5, the LED will not work.

EXAMPLE	CORRECTION
`int LED = 5;`	`int LED = 6;`

Fixing variables that do not match

If a component is not working at all, look carefully at the circuit and see which pins each of the components are sewn to. Make sure the code matches the circuit. If there are any places where circuit and code don't line up, that is a bug.

NOTE: See also the "Incorrect variable initializations" section below.

Address this problem in two different ways. Either re-stitch the circuit or rewrite the code to match the circuit that is sewn. The second option is almost always easier!

Conditions that are always true or false

Symptoms

If something that should happen in the program never happens, or if something happens all the time when it shouldn't, there may be a condition being read as always true or always false because the threshold values are too high or set incorrectly.

TOC

EXAMPLE	CORRECTION

`sensorValue = analogRead(aluminumFoil);`
`Serial.println(sensorValue);`
`delay(100);`
`if (sensorValue < 1024)`
`{`
`  1ightPattern1(2000);`
`}`
	`sensorValue = analogRead(aluminumFoil);`
	`Serial.println(sensorValue);`
	`delay(100);`
	`if (sensorValue < 1000)`
	`{`
	`  lightPattern1(2000);`
	`}`

Here, the **sensorValue** is set too high at **1024,** causing the conditional to *always* be true. If we lower the value to **1000**, another pattern can be called in the 1000 to 1024 range.

Fixing conditions that are always true or false

Look carefully at the conditional statements in the program—**if else** statements and **while** loops. Make sure that the conditions in each statement will actually change as the program progresses or with different sensor readings. Once the problem is identified, revise the conditions to appropriate thresholds so that the program will progress as people interact with the project. Make sure not to get stuck in **while** loops because of missing **i = i + 1;** statements.

NOTE: Extra and/or misplaced semicolons can create similar problems (see below).

Incorrect variable initializations

Symptoms

The wrong component is executing code designed for a different component.

Variables are the foundation of the program. If a variable is given the wrong value at the beginning of the program, it can cause lots of different problems. When pin variables don't match the circuit, it is also a form of incorrect variable initialization (an error covered in "Variables that do not match circuit" earlier). Other initialization problems may be subtler like this example.

EXAMPLE	CORRECTION

`int led1 = 5;`
`int led2 = 6;`
`int led3 = 2;`
`int led4 = A3;`
`int led5 = A4;`
	`int led1 = 5;`
	`int led2 = 6;`
	`int led3 = A2;`
	`int led4 = A4;`
	`int led5 = A3;`

On the ProtoSnap board, the LED pin numbers are already assigned. Many people don't realize that the numbers are not in order on the board, and that the analog pins have an A in front of them. Here, led3 will not turn on at all because pin3 is already pre-programmed to the slider switch. LEDs 4 and 5 will blink out of order because their pin numbers are switched.

Another common issue occurs when copying and pasting code from the one project to the other, because the pin numbers assigned may change from project to project.

TOC

Fixing incorrect variable initializations

Look for the variables related to the areas that are not functioning properly. Compare the variable definitions; identify the ones that are initialized incorrectly, revise, and recompile.

Problems with delay

Symptoms

The delays are too short, too long, missing altogether, or the computer is hanging or crashing. If a delay is too short, it may seem like things that should be happening are not happening. Take a look at the following examples.

EXAMPLE	CORRECTION

`void loop() {`
`  digitalWrite(led, HIGH);`
`  delay(1);`
`  digitalWrite(led, LOW);`
`  delay(1);`
`}`
	`void loop() {`
	`  digitalWrite(led, HIGH);`
	`  delay(1000);`
	`  digitalWrite(led, LOW);`
	`  delay(1000);`
	`}`

In this first example, the delays after the **digitalWrite**(led, **HIGH**); and **digitalWrite**(led, **LOW**); statements are very short. A **delay**(1); means to wait for one millisecond—that's 1/1000 of a second, a tiny amount of time. The LED may look like it's on dimly all of the time, may not activate the LED at all, or do so quickly that it's almost not perceptible.

If the program seems to never respond or respond very slowly, the delay may be too long or a series of delays taking too much time. In the code below, **delay**(20000); will make the project stop and do nothing for 20,000 milliseconds—that's 20 seconds! This means waiting 20 seconds before the project can proceed.

EXAMPLE	CORRECTION

`void lightPattern1() {`
`  digitalWrite(led1, HIGH);`
`  delay(2000);`
`  digitalWrite(led1, LOW);`
`  delay(2000);`
`  digitalWrite(led2, HIGH);`
`  delay(2000);`
`  digitalWrite(led2, LOW);`
`  delay(20000);`
`}`
	`void lightPattern1 () {`
	`  digitalWrite(led1, HIGH);`
	`  delay(2000);`
	`  digitalWrite(led1, LOW);`
	`  delay(2000);`
	`  digitalWrite(led2, HIGH);`
	`  delay(2000);`
	`  digitalWrite(led2, LOW);`
	`  delay(2000);`
	`}`

If the project is freezing or crashing the computer, it may be sending data back to the computer with **Serial.println** statements and overloading the computer with too much data.

TOC

Fixing delay problems

Pay particular attention to the **delay** statements. If a component seems like it's not working, or if entire sections of code take a long time to execute, look for delays that are too short or too long. Remember that programs execute line by line in order and the Circuit Playground can only do one thing at a time, even though it does those things very fast. When a delay is used, the entire program stops until that delay is finished, and then it will move on to the next part of the program. Experiment with changing the delay times. If the computer crashes, add a delay statement to the program right after the **Serial.println** statement. Recompile.

***.

UPLOAD CORRECTION:

```
SERIAL PORT '/DEV/TTY.BLUETOOTH-MODEM' ALREADY IN USE. TRY QUITTING ANY PROGRAMS
THAT…
Binary sketch size: 4,962 bytes (of a 30,720 byte maximum)
processing.app.SerialException: Serial port '/dev/tty.Bluetooth-Modem' already in
use. … uiting any programs that may at processing.app.Serial.(Serial.java:171)
at processing.app.Serial.(Serial.java:77)
```


TOC

MATERIALS

Electronic textile materials and tools

Coin cell battery

A 3-volt battery whose code name is CR2032. This code means that the battery is 20mm in diameter and 3.2mm thick. Flat, round batteries like these are called coin or button cell batteries because of their shape.

Figure 65: Coin cell battery
Image from www.sparkfun.com, CC BY-NC-SA 3.0.

Coin cell battery holder

A sewable battery holder for the CR2032 coin cell battery. Alternatively, a battery pouch could be made out of fabric.

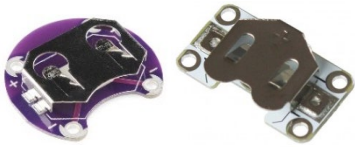


Figure 66: Coin cell battery holder
Images from www.sparkfun.com, CC BY-NC-SA 3.0, & www.kitronik.co.uk.

Conductive thread

A thread capable of carrying electric current. The one we use is spun from fine stainless steel wires, it can be purchased in small quantities on bobbins or larger quantities in spools. Other types of conductive thread include silver-plated threads and gold-wrapped embroidery threads. Be aware that thread with silver in it will tarnish over time and causing conductivity issues.

Figure 67: Conductive thread
Images from www.adafruit.com.

LED

A small sewable LED (light-emitting diode), or a standard "through hole" LEDs. Simply twist their legs into sewable loops. To help with remembering which leg is positive and which leg is negative, always do the same pattern for positive (square or zigzag) and the same pattern for negative (simple twisted loop).

Figure 68: LED
Images from www.sparkfun.com.

Materials section based on:

Buechley, L., Qiu, K., Goldfein, J., & de Boer, S. (2013). *Sew electric: A collection of DIY projects that combine fabric, electronics, and sewing* [online version]. Retrieved from http://sewelectric.org/references/glossary/.

TOC

USB Cables

A USB 2.0 A to Micro-B Cable connects the Circuit Playground to the computer. This is another common cable that we often have around the house from other electronics.

Be aware that USB cables often malfunction or are made slightly differently. Some do not have the same data transferring capacity as the ones we use in e-textiles. If the computer does not recognize the Circuit Playground, go to the Troubleshooting: Connecting the Circuit Playground and the Computer section, and see if the cable might be the issue.

Crafting materials and tools

Glue

Any standard craft glue will work well for sealing knots, including fabric glue or Elmers® glue. Use fabric glue or felt glue to attach one piece of fabric to another. The larger the pieces of fabric, the less effective the glue will be.

Large-eyed needle

"Chenille" needles sized 18-24 are very easy to thread and fit through the holes in Circuit Playground pieces. More experienced sewers may want to use a smaller needle, but be careful, smaller needles are much harder to thread and may be sharper. Larger needles may leave larger holes in the fabric.

Seam ripper

A small tool used for ripping/cutting out stitches. There is a protective ball on one end to help prevent jabbing a person, or accidently cutting into the material instead of the stitch.

Polyester filling

The material that's inside pillows and stuffed animals. Have an extra pillow? Scavenge the stuffing to make a plush e-textile toy! When buying stuffing, go for the synthetic kind. Do not use pellets or feathers.

Felt sheets

These thick non-stretch fabrics are easy to sew and come in many colors and can be found at any craft store or Walmart. They are made out of spun plastic, so caution should be taken if attempting to use with a hot iron. Normally a sheet of paper between the felt and the iron is enough to use Iron-On adhesive without melting the felt. Other fabrics that don't unravel include wool felt and polar fleece.

Aluminum foil

Standard aluminum foil, heavy-duty. Do not try to use the non-stick variety.

Iron-on adhesive

Thermoweb®'s Ultra Hold Heat-n-Bond adhesive. We'll need the sheet, not the tape. This is machine-washable, and should not gum up the sewing needles.

Figure 69: Heat-n-Bond adhesive
Image from www.craftsy.com.

TOC

IDEAS & INSPIRATION

DIY Sensor Ideas

Electronic Textile Design Ideas

TOC

Do-It-Yourself Sensors

The tactile sensor we need for the Human Sensor Project is made of aluminum foil. Use an iron to first line the foil with adhesive. Then measure out and cut the patches out for the project, peel off the paper backing from the adhesive sheet, stick the patches onto the project with the iron (if the project is flat) or a crafting iron (if the project is three-dimensional), and then sew them with conductive thread to connect them to the Circuit Playground.[19] You can also view a video tutorial of how to make a large scale production of conductive patch material to be cut up into conductive patches as needed.

Making touch sensor patches

Materials:

- Aluminum Foil (heavy-duty, do not use non-stick)
- Heat 'n' Bond Iron-on Adhesive (double-sided, ultra-hold)
- An Iron and an Ironing Board
- Cover for the Ironing Board (can be a piece of scrap fabric or paper)
- Scissors (not sewing)
- Writing Implement (optional)
- Crafting Iron (optional)

Making sensor patches:

Step 1: Read the instructions on the Iron-on Adhesive packaging to find the correct temperature setting for the iron. Do NOT use the steam setting!

Step 2: Pre-heat the iron and cover the ironing board with paper or scrap fabric. This is to keep the melted adhesive from sticking to the ironing board.

Step 3: Tear off a piece of foil that matches the width of the iron-on interface. It's best to keep the foil in large rectangular pieces at this time. These sheets can be provided to students who can trace their designs on the paper side, cut them out, and then iron on the smaller pieces late. Place the aluminum foil on the ironing board, shiny-side down if we want the sensors to be shiny, matte-side down if we want the sensors not to be shiny.

Step 4: Cut off a matching piece of iron-on adhesive. It can be a little bit smaller than the foil, but not any bigger than the piece of foil. Place it rough and shiny (adhesive) side down onto the foil.

[19] Buechley, L., Qiu, K., Goldfein, J., & de Boer, S. (2013). *Sew electric: A collection of DIY projects that combine fabric, electronics, and sewing* [online version]. Retrieved from www.sewelectric.org/diy-projects/interactive-stuffed-monster/give-your-monster-a-sense-of-touch/.

TOC

Step 5: Place the iron on top of the adhesive sheet, count to five (or see instructions on the adhesive product). Move the iron to another spot, iron each spot for five seconds. When we're done, the adhesive should be fused flat against the foil. If we missed any spots, iron those spots again. Make sure that the adhesive is firmly attached by peeling up a small corner of the paper. It should peel away easily and a clear layer adhesive should be stuck to the foil. Just peel up enough paper to check on the adhesive – don't remove the paper yet.

Step 6: Turn off the iron if it won't be used anymore!

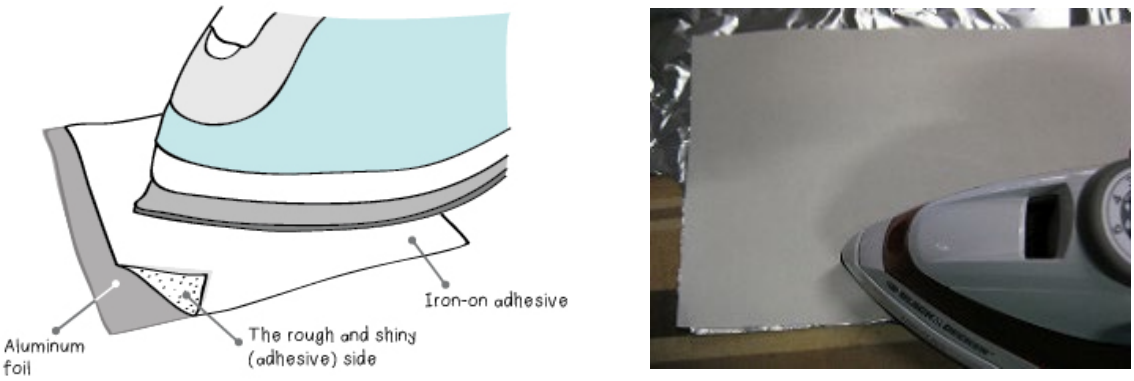


Figure 70: Using an iron
Images from www.sewelectric.org, CC BY-NC-SA 3.0, & www.kobakant.at.

NOTE: The adhesive melts with heat. We use it alumium foil to the adhesive, and later to the fabric. If one makes a mistake in a project, it *can* be removed by ironing it again and pulling it off while still very hot. This will leave behind some sticky residue but at least allows for emergency adjustment of patches when necessary.

Attaching the touch sensor to the project

Now measure out and design the sensor patches. The sensors can better gather information when they are sized at least two inches around (bigger patches = greater surface area = wider sensor range). Circles and square patches work really well and are simpler to cut out than custom-designed patches. If we need two mirror-image patches (like a left hand and a right hand), draw the outline onto the paper (adhesive-side).

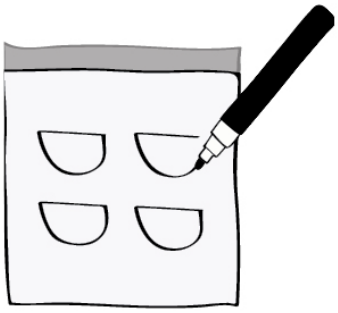


Figure 71: Designing sensor patches
Image from www.sewelectric.org, CC BY-NC-SA 3.0.

Keep in mind that we want the foil-side to be on the front of the project. A shortcut is to fold the foil and adhesive in half and cut two patches out at the same time. This works if the patches are mirror-image.

Tip: Sketch out the patch on a blank piece of plain paper, cut it out, then *flip it upside down,* and trace it onto the paper side of the aluminum foil. Then cut out the conductive patch. It's important to flip it upside down because the foil side will be up (not the paper side) in the final product. [If this sounds confusing, play around with it!]

TOC

Pre-heat the iron. Crafting irons have smaller plates and may be easier for ironing surfaces that are not flat. Check to make sure the patches fit on the project. Peel the paper off, place the adhesive side down onto the fabric, and make sure again that the aluminum foil faces out. Use the five-second guide and iron the patches onto the fabric spot by spot, holding the iron on it for five counts.

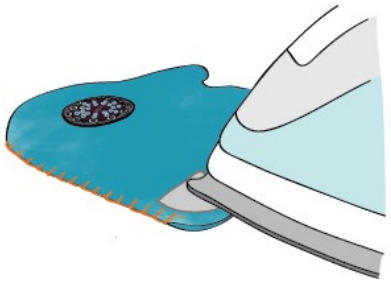


Figure 72: Ironing patch
Image from www.sewelectric.org, CC BY-NC-SA 3.0.

WARNING: Irons are hot! They will burn you if you touch the metal plate.

WARNING: The adhesive might ooze out and stick to the iron (or on the project, someone's skin or clothes). Use rubbing alcohol to remove.

WARNING: Be careful when ironing patches to felt. Felt is made of spun plastic and the project will melt at high temperatures!

TOC

Electronic Textiles Design Ideas

Makers, even the most experienced ones, often have difficulty coming up with ideas. While the ECS curriculum provides some examples of projects, we find that seeing a broad database of images sometimes gets our own creative juices to flow. Below are some inspiring ideas of finished products for all of the e-textiles activities in the ECS curriculum.

Activity 1: Electronic greeting card

There are two design options for this project: We can use the copper tape as part of the visible design (left) or overlay the copper circuitry with another piece of paper (right).[20]

Figure 73: The copper tape as part of a visible design
Image from www.crowdsupply.com.

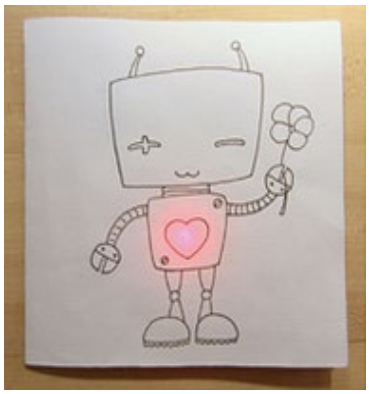


Figure 74: Paper overlay—the circuitry is covered up
Images from www.crowdsupply.com and http://scraptime.ca/.

The simpler option is to design a paper cover to go over the circuit, like in Figure 74, and use the rectangular circuit template provided by Chibitronics (Figure 75).

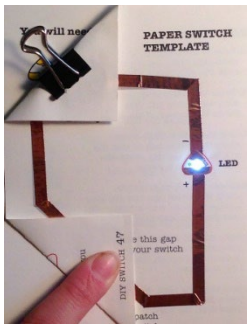


Figure 75: Simple template provided by Chibitronics
Image from www.chibitronics.com.

[20] Qi, J. (2014). Circuit stickers tutorial. *Chibitronics, PTE LTD.* Retrieved from www.chibitronics.com.

TOC

Below are a few more greeting card ideas using a circuit with one light. Notice that the artwork can be minimal and unsophisticated, and it can still be effective and fun for the recipient to receive this unique greeting card. We can also poke a small hole in the paper overlay to let the head of the LED stick through.

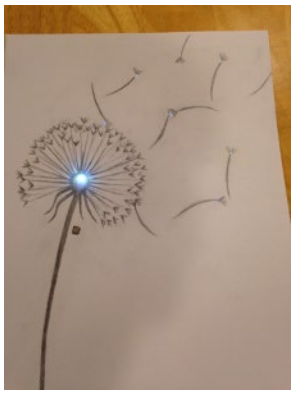
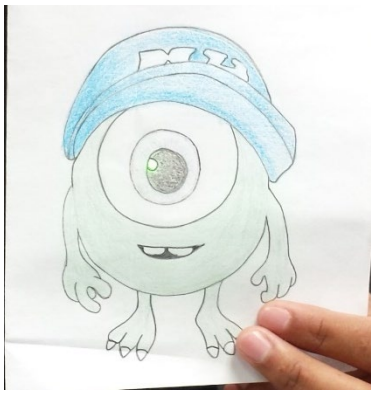


Figure 76: Greeting card idea
Image from www.chibitronics.com.



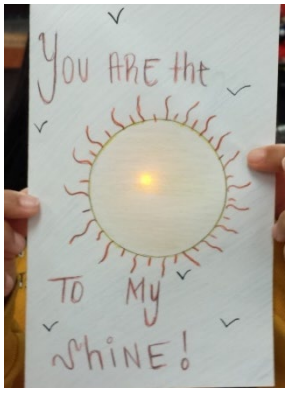


Figure 77: ECS greeting cards
Images from ECS Pilot Study, Year 1

Using the tape as a visible part of the design presents many additional challenges:

- The copper tape does not curve or turn corners very well. The tape may get crinkled
- The negative and positive traces must not touch one another
- Scotch tape will most likely have to be used to ensure the copper tape connects conductive side to conductive side (the side of the copper tape that is not adhesive) at junctions. The plastic tape may have an unsightly effect on the project
- The project will use up much more copper tape than the others.

But it is possible to do it, and it may be worth the effort if it's planned well. Here is an example:

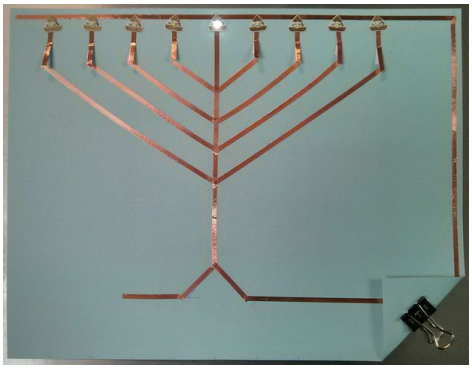
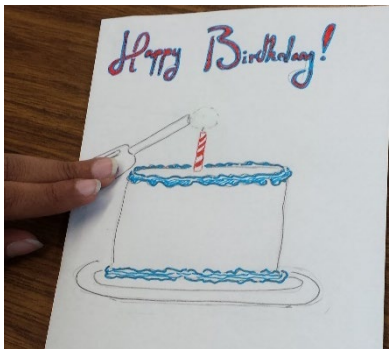


Figure 78: Visible tape
Image from www.instructables.com: megapix.

TOC



We can also consider incorporating the switch into the design. This student from the first ECS pilot study made a birthday card (left). The note at the bottom of the card instructs the card recipient to touch the lighter to "light the candle."

Figure 79: Switch as part of design
Image from ECS Pilot Study, Year 1

Activity 2: Electronic wristband

The aesthetic drawing below is a great template for the electronic wristband (Figure 80). The lights are in a parallel circuit, and the metal snaps serve as a switch because the negative line does not connect to the power source unless the bottom two snaps are connected. Use this as a guide for designing the wristband. We can, however, make the lines straight across (not wavy like shown here), or move the LEDs so they're not all evenly spaced and aligned across the middle like in the drawing. We can also choose whether the battery holder should be on the front or the back side of the fabric.[21]

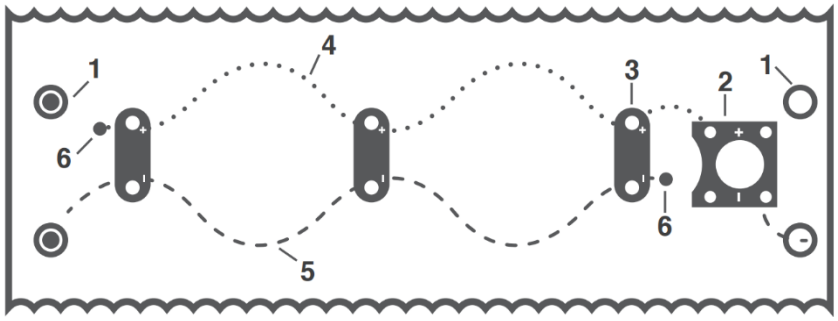


Figure 80: Wristband sample drawing
Image from www.itls.usu.edu: Light-Up Bracelet Student Guide

In the three examples on the following page, the circuitry is mostly covered up by pieces of fabric, with the LEDs sticking up through little cut-outs in the fabric (Figure 81 and 82). The batteries were also sewn in on the back sides of the fabric or on the bottom of the wristband where they are not visible in these photos.

[21] Strommer, A. & Fields, D. (n.d.). Light-up bracelet student guide. *Utah State University*. Retrieved from https://itls.usu.edu/files/projects/ETextiles_Bracelet_Guide.pdf.

TOC

Figure 81: Designs with fabric overlay
Images from http://www.instructables.com: bitwiseOwl, & www.darcyneal.com.

We don't have to be limited to wristbands. This ECS student made a collar for her dog:

Figure 82: E-textile dog collar
Image from ECS Pilot Study, Year 1

As with the Electronic Greeting Card activity, making the circuitry visible on the front is more challenging. But it can be done!! The next page shows four ideas where the stitching is part of the aesthetic design:

TOC

Image from www.kpeppler.com.

Image from www.highlowtech.org.

Image from www.itls.usu.edu: Light-Up Bracelet Student Guide

Image from www.primaryinspired.net

Figure 83: Four designs with visible stitching

Once we choose the fabric and create our circuit diagram, be sure to include space for the battery and the switches before cutting the fabric. Taping the components down with Scotch tape can help us figure out how much material and space we will need. We can also use a pencil or a fabric pen to draw where the stitches will go, directly on the fabric.

The snaps are really tricky. They are distributed already connected in pairs, and a "male" side and a "female" side are necessary to make up a pair. Disconnect them and tape them onto the fabric in their respective places until we need to sew them, to avoid them getting lost or mixed up. Try the wristband on a few times while crafting to make sure the snaps are oriented correctly.

Activity 3: Collaborative Mural project

This project is collaborative in several ways. First, we emphasize designing in pairs, pair crafting, and pair programming – in other words, two people working on one project concurrently. Then the projects are brought together into a larger project that has a cohesive message.

When working in groups, it is tempting for the sewing experts to do all the sewing and all the programming experts to do all the coding. The purpose of the pair-work is for division of labor to be as equal as possible, with each participant doing half of the crafting and half of the programming, so that no one person is doing everything. While it is natural for each person to have a preference for

TOC

one aspect of the work than the others, we want to encourage partners to share their expertise and tips with their partners, rather than take it and just do it all themselves. That way, everyone learns!

Think about a sign for the classroom, the computer science program, or for the school. Does the school band need a banner for the local parade? Could the classroom or the front office use a new welcome sign? Does a demonstration booth or fundraising table need a decorative? Honing in on the "what" will determine the materials we need and how to divvy up the tasks. Think of a simple phrase to spell out, and assign students to make their letter come alive in a special way. Be sure to facilitate discussions with the design partner, the group at large, and other school community stakeholders. Remember that each individual piece will have its own battery and two switches for interactivity with the audience.

Here are some possible project ideas. This mural (Figure 84) was created by three adults in just a few hours and utilizes only our basic e-textile materials (felt, Circuit Playground, LEDs, switches, etc.).

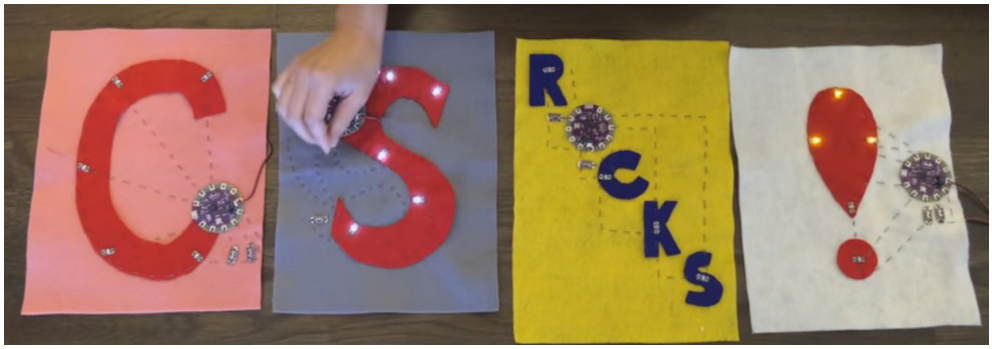


Figure 84: Computer Science rocks!
Image by Deborah Fields.

This is a video of that project in action: http://bit.ly/CSrocks-vid.

The following series was created by students at a magnet school. The project was to create an interactive mural that spells out the name of the school (not provided in order here, to protect the anonymity of the school). Before it was even created, school administrators had determined that the project would be permanently installed in a high traffic area where students, staff, and visitors to the school would interact with it. The project involved multiple layers of collaboration. First, each "letter" panel was designed by a student in the fine arts program and their art was printed onto canvas. That student had multiple design meetings with the two computer science students assigned to create the light patterns. Together, they determined where the Circuit Playground, LEDs, switches, etc. would go and how the piece would interact with the people who would walk by. Before the project was installed, administrators of the school were invited to view and play with the completed art.

TOC

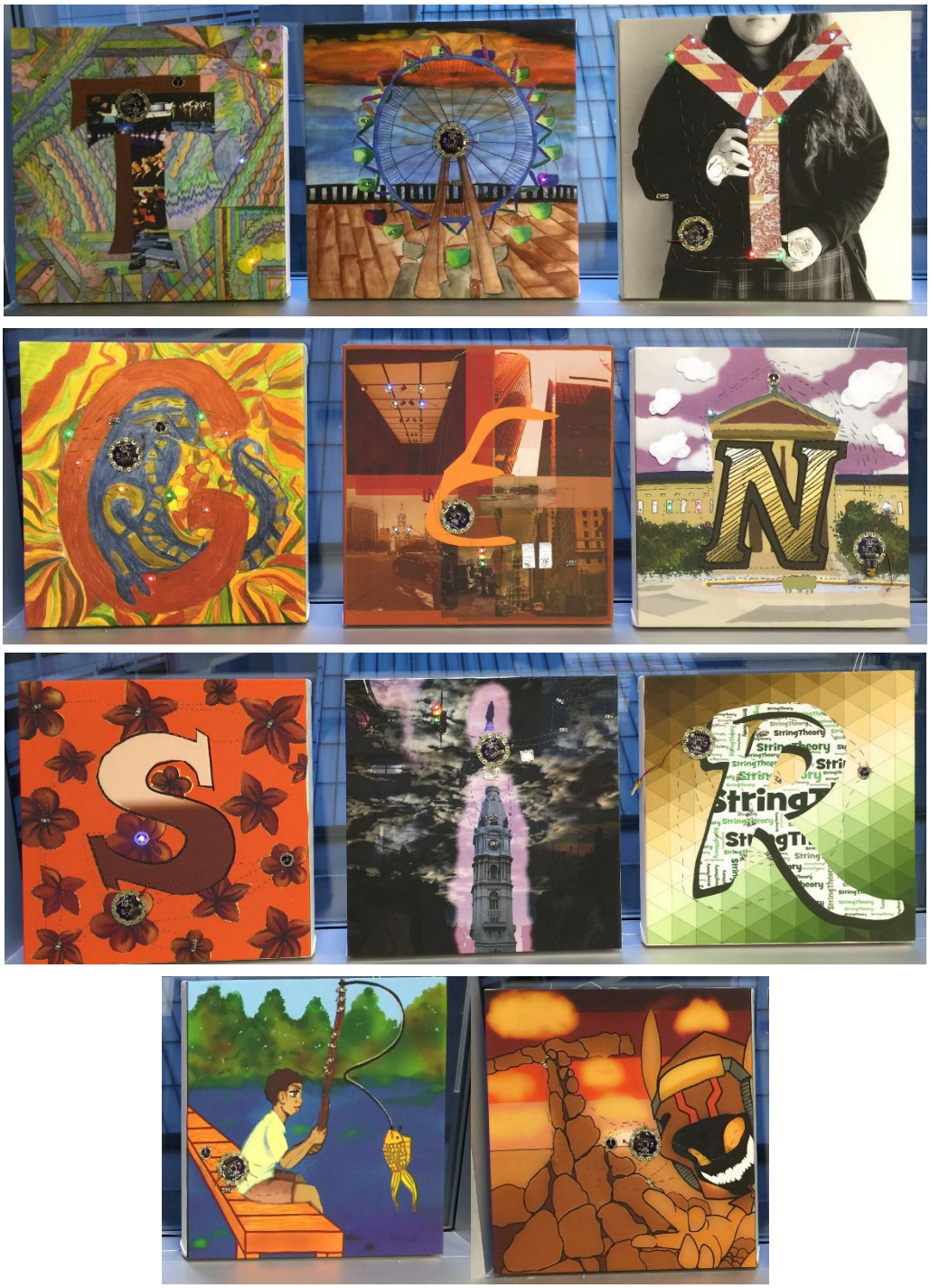


Figure 85: Art and Computer Science student collaboration
Images from University of Pennsylvania Pilot Study.

TOC

Figure 86: Mural Example from an ECS class with an animation theme.
Images from UCLA Pilot Study.

TOC

Activity 4: Human Sensor Project

The Human Sensor Project also requires interaction with an operator (user) in order to function. Instead of a simple on-off switch, the HSP uses a sensor to read a range of inputs from the user. We can program the Circuit Playground to function differently depending on the sensor reading. In this case, touch sensors will gauge how conductive the operator is. Each of these projects are designed to produce different lighting effects depending on how much the user is touching the sensor patch; the effects change based on how much surface area contact there is (e.g., no contact vs one finger touch, pinch with two fingers and thumb vs. whole hand squeeze).

Brainstorm about making something that'll be fun to touch and interact with. What should people touch or squeeze? Should there be only one human interacting with the computer, or multiple people? Should multiple humans interact with one another *and* the project?

Here are some ideas. Below is a pillow with a dinosaur and two sensor patches (Figure 87). As the operator applies more pressure to the touch sensors, the dinosaur lights up in different ways.

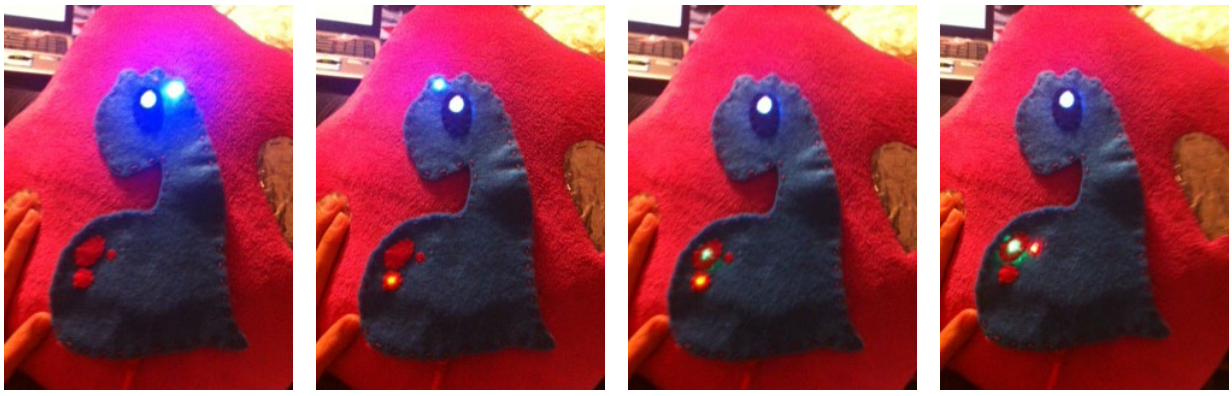


Figure 87: Dinosaur pillow with four behaviors
Images from http://blog.usu.edu/: Craft Technologies 2013.

This is Clyde (below), a stuffed human sensor monster created by Janell Amely. The monster's appendages are lined with aluminum foil. See Clyde in action on YouTube: http://bit.ly/Clyde-in-action.

Figure 88: Clyde, a stuffed monster
Image by Janell Amely.

TOC

NOTE: As we may have experienced in the wristband project (and if we've made clothing before), it's really tricky to think three dimensionally, to space things correctly, to remember where the fronts and backs are, to figure out which parts to complete first, etc. With a stuffed project, we have to be additionally careful that the Circuit Playground switch and battery are accessible from the outside, and that connections are checked and double-checked before stuffing the project and sewing all the edges up. Proceed with stuffed projects only after creating very accurate and detailed circuit drawings of both the front and back (and other perspectives, if applicable).

We recommend for beginning crafters to modify existing wearable articles of clothing, or to create something that's flat (2-dimensional), like these:

Figure 89: Fuzzy hat with ear muff sensors
Image from http://blog.usu.edu/: Craft Technologies.

Figure 90: Children's blanket
Image from http://blog.usu.edu/: Craft Technologies 2013.

Figure 91: "L"ED Jacket
Images from http://blog.usu.edu/: Craft Technologies.

Mark Stevenson created a "Get Along Mat" for his children. He told his kids that the mat requires two people to hold hands while touching the foil sensors. The firmer the contact is with the sensors, the more LEDs light up. He remarked that he has heard his oldest son threatening his younger brother to play nice or he would ask Dad to pull out the "Get Along Mat." See it in action on YouTube here: http://bit.ly/get-along-mat.

TOC

Figure 92: "Get Along Mat"
Image from Mark Stevenson

Below is a dog halter that Erin Anderson modified with LEDs around the collar. The sensor patch is on the wristloop held by the human, not on the dog. When the dog pulls, it forces the operator to grip the band tighter, the LEDs create different lighting patterns. The maker said the dog broke it the first time he wore it — he doesn't like halters!). See an explanation for how it works here: http://bit.ly/dog-halter.

Figure 93: Light-up dog halter
Image by Erin Anderson.

NOTE: As a reminder, always consider the audience or user, as well as the intended longevity of the artifact, when designing the project. The materials we use in e-textiles—felt, foil sensors, conductive thread, etc.—are all very fragile. Even Clyde, the stuffed monster above, had to be patched up with conductive fabric after the foil wore down.

TOC

GLOSSARY

This section contains definitions of key technical words that are used in this resource guide or in the ECS curriculum. These short descriptions are not meant to be a rubric for assessing student knowledge.

abstraction: the computer's process of creating pieces of code, so that a programmer does not need to understand how the code works to use it in a program.

analog: in electronics, a signal that conveys information through a continuously changing value. Analog signals are different from digital signals, which convey information through discrete changes in values. An analog signal looks like a wave moving up and down. See also **digital**.

array: a list or table in code. Arrays provide a way to store an ordered collection of data in our program. The entries in an array are accessed using numbers in square brackets. The access numbers begin with zero.

C (programming language): a general-purpose programming language developed in the late 1960s by Dennis Ritchie, a researcher at AT&T Bell Labs. All Arduino programs are written in C. However, the Arduino environment has many features that aren't part of standard C, including built-in procedures like digitalWrite and delay, so the Arduino language can be said to be a dialect (or version) of C. C is one of the most widely used programming languages in the world.

call (a procedure): calling a procedure means that we use the procedure in the main part of our program. For example, we call Arduino's built-in delay procedure when we include the statement **delay(1000);** in the loop part of our program.

capacitance: a material's ability to store electric charge.

capacitive sensor: a sensor based on changing capacitance, that changes its capacitance in response to a stimulus, often touch.

circuit: a network of electrical devices. It is generally a closed loop that includes a power supply and other electrical elements like LEDs and switches. The loop structure enables the flow of electric current from (+) to (-).

code or **source code**: a collection of instructions that will be carried out by a computer (or Circuit Playground) and that are written in a programming language. Any piece of an Arduino program can be called code. See also **program**.

This material was adapted from Sew Electric and customized for use with the ECS curriculum. See:

Buechley, L., Qiu, K., Goldfein, J., & de Boer, S. (2013). *Sew electric: A collection of DIY projects that combine fabric, electronics, and sewing* [online version]. Retrieved from http://sewelectric.org/references/glossary/.

TOC

comment: a piece of text in a program that is ignored by the compiler and the computer (or Circuit Playground) that executes the code. In the C programming language there are two kinds of comments: comments that extend for several lines (like paragraphs) and comments that are written on a single line. To create a comment that spans multiple lines, begin the comment with /* and end it with */. To create a comment that is only on one line, begin it with //. Comments are shown in a greyish brown in the Arduino environment.

comment out (code): to turn a piece of code into a comment. Commenting out a piece of code allows us to temporarily prevent that code from executing—since comments are ignored by the compiler—while keeping the text of the code in our program. To comment out a single line of code, add two slash characters // to the beginning of the line. To comment out a larger piece of code, add /* characters to the beginning of the section we want to comment and */ to the end of the section. The commented-out section turns grey in the Arduino window. The statement **delay(1000)**; is commented out when two slash characters are placed in front of it like so: **//delay(1000);** See also **comment**.

communication speed: the speed at which one computational or electronic device communicates with another. Communication speed is measured in bits per second. 9600 bits per second ("baud") is the standard communication speed for most Arduino applications. Before sending data from a Circuit Playground to a computer in an Arduino program, we must specify the communication speed using the Serial.begin procedure. For example, the statement **Serial.begin**(9600); sets the Circuit Playground's communication speed to 9600 bits per second. See also **serial port**.

compile: to translate a program, written in a programming language like C, into a machine-readable code like hex code. In the Arduino, clicking on the check mark icon in the toolbar compiles C code that we wrote into hex code that a Circuit Playground can understand.

compile correction: an error in a program that is detected when software attempts to compile the program. Compile errors are often syntax errors like misspellings or missing semi-colons. For example, if the word "delay" is misspelled in the statement dellay(1000);, code containing this statement will generate a compile error.

condition: a statement that is always either true or false. Conditions are often comparisons using less than (<), greater than (>), and equal to (==) operators. For example, the condition x < 10 is true when x is less than 10 and false when x is 10 or greater. Conditions are an important part of conditional statements, where they appear in parentheses. In the line **if (i < 10)**, **(i < 10)** is the condition. See also **conditional statement**.

conditional statement: a block of code that does one thing if a condition is true and another thing if the condition is false. if else statements and while loops are examples of conditional statements. See also **condition**.

conductive: a conductive material is one that an electrical current flows through easily. Metals like copper, silver, and aluminum are highly conductive. Conductive materials include the tabs on the Circuit Playground components and the metallic thread used to sew these components together. The opposite of conductive is insulating. Insulating materials—like plastic, glass, and fabric—resist the flow of electricity. Electrical current does not flow through insulating materials.

TOC

constant: a named variable in a program whose value never changes. Constants are declared like variables with a type and a name. However, constants require an additional code word, const, at the beginning of the statement.

digital: information represented using discrete values. In computing, information represented using 1s and 0s. In electronics, a signal that conveys information through discrete values (**HIGH**/**LOW**) instead of continuously varying values. A switch is an example of a digital sensor since it conveys information in a discrete manner; it is either open or closed. Digital signals are different from analog signals, which convey information through continuous changes in values. A digital signal looks square since it is only on or off for a period of time. See also **analog**.

electric current: the flow of electricity through a circuit. Current travels from the (+) side of a battery (power), through the components connected in the circuit, and back to the (-) side of the battery (ground). Electric current only flows through a circuit if the path is complete, that is, if there are no breaks in the circuit. Electric current is measured in amps.

electronic textiles or **e-textiles**: fabrics that include soft electrical circuitry and embedded electronics like sensors, lights, motors, and small computers. Designers of e-textiles strive to keep things soft by using new materials like conductive thread, conductive fabric, and flexible circuit boards.

energy: electrical energy is the ability of a power supply to run a circuit over time, to light up an LED, or make sounds with a speaker. The amount of energy stored in a battery is equal to its amp-hour rating multiplied by its voltage rating. Energy is measured in watt-hours (Wh). For example, a 3-volt battery with an amp-hour rating of .25 amp-hours stores .75 watt-hours of energy.

execute (a program): (also **run**) the act of carrying out the instructions specified by a program. A computer—like the Circuit Playground Arduino—executes programs line-by-line in the exact order in which they are written.

frequency: the speed or pitch of a sound wave. Sound is created by vibrations of molecules in the air. When the molecules vibrate very quickly—at a high frequency—we hear a high note; when they vibrate more slowly—at a low frequency—we hear a low note. Frequency is measure in pulses per second or Hertz (Hz).

ground (-): the negative terminal of a battery or other power supply in a circuit; 0 volts. Also, any part of a circuit that is at 0 volts. Ground is the reference point in a circuit from which all other voltages are measured. The color black is used to denote ground in circuit diagrams and drawings. In Arduino code, ground is referred to as **LOW**. See also **low** and **power**.

high or **HIGH**: in Arduino code, the term used to refer to power (+) in electrical circuits. Power, (+), and **HIGH** refer to the positive terminal of a battery or other power supply in a circuit. See also **power** and **low**.

initialize (a variable): a variable is initialized when a value is assigned to it for the first time. For example, the statement **int led = 13;** declares a variable called led and initializes it to 13. The statement assigns the value of 13 to the variable led.

TOC

input (device): an electrical component that gathers information from the world. This information could include how hard a sensor is being pressed, the current temperature, or the ambient sound level is. Input devices include switches, touch sensors, thermometers, cameras, and microphones. All sensors are inputs. In Arduino, information is collected from inputs with **digitalRead** and **analogRead** statements. See also **read** and **output** (device).

input (to a procedure) or **input variable**: information required by a procedure. Usually a number supplied in parentheses after the procedure name in a procedure call. For example, in the statement **delay(1000);**, the number 1000 is the input. Input variables help programmers write procedures that are applicable to a wide range of situations. Inputs enable us to carry out the same basic set of instructions or statements, but with different values. The built-in procedure delay has an input that lets us delay for different amounts of time in the programs. When we call delay with an input of 1000, our program pauses for one second. A delay with an input of 100 pauses for 1/10 of a second.

int or **int**: a data type used in Arduino programs used to declare integer (i.e. whole number) variables. The most common variables and numbers are of type int. For example, the statement **int led = 13;** declares a variable called led of type int. See also **type**.

LED: short for light-emitting diode. LEDs contain an electroluminescent material—a material that glows when electrical current flows through it. LEDs are polarized. That is, they have a (+) and a (-) side and will only light up when current flows from their (+) to (-) side. If we attach an LED backwards in the circuit, it will not work. LEDs are more efficient than most other light sources. That is, they produce more light with less energy than most other types of lights. See also **polarity**.

logical correction: an error that occurs when our code compiles and uploads but doesn't behave the way we want it to. These errors are the trickiest to find and fix because the computer doesn't give us any feedback about what might be causing problems, like it does with compile and upload errors.

low or **LOW**: in Arduino code, the term used to refer to ground (-) in electrical circuits. Ground, (-), and **LOW** refer to the negative terminal of a battery or other power supply in a circuit. **LOW** is always 0 volts. See also **ground** (-) and **high**.

memory: where an Arduino program is stored once it's uploaded to a Circuit Playground. Once a program is stored in the Circuit Playground's memory, the Circuit Playground can run the program independently of the computer.

microcontroller: a small computer chip that stores and execute programs and controls electronics. Microcontrollers, like most computers, have a memory that is used to store programs and program data, and a processor that is used to interpret and execute programs. Microcontrollers also have pins that can be used to control input and output devices. When input and output devices like sensors and LEDs are attached to the pins, the microcontroller can read electrical signals from the inputs and send electrical signals out to the outputs.

TOC

milliseconds: 1/1000 of a second. 1000 milliseconds (1000 ms) is one second. The input to the Arduino procedure delay is in milliseconds. **delay(1000);** pauses program execution for one second, **delay(100);** pauses program execution for 1/10 of a second, and so on.

open source: a term used to describe a program whose source code is publicly available for us to use, examine, and modify. The sharing of open source software, also known as free software can enable people to collaborate on projects by helping them extend and expand on each other's work. We are making use of open source software and open source hardware throughout this curriculum. Arduino is open source and the designs of the Circuit Playground boards are also open source.

output (device): an electrical component that takes action—does something—in the world. Actions could include lighting up, moving, making sound, or changing shape. Output devices include lights, motors, speakers, and display screens. In Arduino, outputs are controlled by **digitalWrite** statements. See also **write** and **input** (device).

parallel circuit: components in a parallel circuit have all of their (+) sides connected together and all of their (-) sides connected together, so all of the components receive the same voltage.

pin: part of a microcontroller that can attach to and control an input or output device. Each microcontroller pin can control either an input device, like a switch, or an output device, like an LED. The pins on a microcontroller look like tiny legs coming out of the controller's black square body. On the Circuit Playground boards, microcontroller pins are connected to sewable tabs and snaps. When an input or output device is attached to a tab or snap, the microcontroller can control that component. Can also be called a **tab** or **microcontroller**.

polarity: electrical orientation or direction. Electricity flows in one direction in a circuit, from (+) to (-). Due to this property, electrical circuits are said to be polarized. Electrical components with polarity will only work properly when electrical current flows through them in a particular direction, from their (+) to (-) side. LEDs are examples of components with polarity.

power (+): the positive terminal of a battery or other power supply in a circuit. Generally, the highest possible voltage in a circuit. The color red denotes power in circuit diagrams and drawings. Note: Different circuits may have different power (+) voltages. For example, in a circuit that uses a 3-volt battery, power is +3 volts. In a circuit that uses a 3.7-volt battery, power is +3.7 volts. In Arduino code, power is referred to as **HIGH**. See also **high** and **ground**.

procedure: a block of code that is given a unique name. A procedure may have one or more inputs and it may return a value. When a procedure is called, the program jumps to the place in the program where the procedure is defined, executes the block of code that makes up the body of the procedure, and then jumps back to the point right after the procedure was called in the program. The Arduino language has a library of built-in procedures like **delay**, **digitalWrite**, and **analogRead**. We can also define our own procedures. See also **input** (to a procedure) and **return**.

TOC

program: a set of instructions to be carried out by a computer (or Circuit Playground). A program is written in a programming language. Programs can also be called pieces of code. A program does its work when a computer runs or executes its instructions by following them in order. See also **code**.

programming language: a language that enables people to write instructions for computers. Programming languages generally have limited vocabularies and very strict formatting rules. This enables them to be read and understood by machines. There are many different programming languages like Python, Java, C++, and Scheme. Arduino programs are written in the C programming language.

read (from a pin): the act of gathering information from an input device. Information is collected via an input device attached to a pin on a Circuit Playground (or other microcontroller). In Arduino, information is read from a pin with the statements digitalRead and analogRead. **digitalRead**(pin); tells us whether the pin is **HIGH** or **LOW**. **analogRead**(pin); measures the voltage level of the pin and gives us a number between 0 and 1023 that corresponds to the voltage. See also **write** and **input** (device).

resistive sensor: a sensor based on changing resistance, that changes its electrical resistance in response to a stimulus.

return: a procedure is said to return when it finishes executing. Some procedures return a value. That is, the procedure finishes executing and provides the results of its execution back to the program that called it. For example, the procedure **analogRead**(pin); returns the value it read from the input pin. See also **procedure**.

run: see **execute**.

running stitch: the most basic stitch in hand sewing. Also called a straight stitch. This stitch is created by passing a needle and thread up and down through a piece of fabric along a line. A good running stitch consists of neat, even stitches of about 1/4" (6mm) in length.

sensor: an electrical component that gathers information from the world, for example: how hard a touch sensor is being pressed, what the current temperature is, or what the ambient sound level is. Sensors include touch sensors, thermometers, cameras, and microphones. All sensors are inputs. In Arduino, information is collected from sensors with the **analogRead** statement. See also **read** and **input** (device).

serial port: the communication channel through which a computer communicates with a Circuit Playground and vice versa. The serial port connection—the USB attachment between the Circuit Playground and the computer—allows Arduino to upload programs to the Circuit Playground and allows the Circuit Playground to send information back to the computer with Serial.println and Serial.print statements. See also **communication speed**.

short circuit or **short**: the direct electrical connection of a power supply's (+) and (-) sides. When a short circuit occurs, the circuit's power supply releases a tremendous burst of energy. A short circuit can ruin a battery and can electrocute or burn us if the power supply is powerful enough. The batteries we're using for the projects probably won't shock or burn us even if we create a

TOC

short circuit. But, if we do connect the project's (+) and (-) sides or connect traces that should not be touching, the project may not work and we may quickly ruin the battery.

statements: computer sentences; lines of code written in a programming language that tell a computer to do something. Simple statements in Arduino end with a semicolon the way that English sentences end with a period. **digitalWrite(led, HIGH);** is a simple statement. More complex statements can span several lines. Examples of complex statements include if statements and while loops.

switch: a circuit component that is always in one of two states: open (disconnected) or closed (connected). In a simple circuit, the flow of electricity through a circuit is stopped when the switch is open and restored when the switch is closed.

syntax: the rules that define the structure of a programming language. These are the spelling, punctuation, capitalization, and formatting rules we must follow when we're writing a program. Different programming languages have different syntaxes.

syntax correction: an error in a program that happens when we do not follow the programming language's syntax. See also **syntax** and **compile error**.

threshold: a cutoff value used in a program such that one behavior happens if a variable has a value below the threshold and a different behavior happens in the variable has a value above the threshold. Thresholds are often used in conditional statements of the form **if (variable < threshold) or if (variable > threshold)**.

trace: a conductive connection between two components in a circuit.

type: computer programs handle different kinds of information including whole numbers, decimal numbers, text, and images. Before a program can manipulate a piece of data, it needs to know what type of data it is dealing with—whether it's an image, a piece of text, etc. In most programming languages, each variable used in a program must have a specified type when it is first declared. Almost all of the variables we use are of type integer or int. They are whole numbers. In Arduino, integer variables are declared with a statement of the form int variableName; or intvariableName=#;. In these statements, int specifies the variables' type. See also **variable declaration**, **int** and **integer**.

upload: the act of sending code that has been compiled and converted into hex code from a computer to a Circuit Playground.

variable: a named location in a program's memory that can store values. Variables give names to information and elements used in our program. They make programs easier to write, understand, and modify. Variables are generally listed at the beginning of an Arduino program. This list is analogous to the list of ingredients at the beginning of a cooking recipe. Every Arduino variable has a type that is specified when it is declared. See also **variable declaration**, **variable initialization** and **type**.

TOC

variable declaration: a statement introducing (or "declaring") a variable and its type. For instance, the statement **int myVariable;** declares a variable named **myVariable** of type **int**, short for integer. All variables in a program must be declared before they can be used. Variables are often declared and initialized in a single statement. The statement **int myVariable=13;**, for example, both declares a variable called **myVariable** and initializes it to the value 13. See also **variable**, **variable initialization** and **type**.

variable initialization: a statement that gives a variable a value for the first time. For example, the statement **int myVariable=13;** declares a variable called myVariable and initializes it to the value 13. Variables are often declared and initialized in the same statement, but not always. The statement **int myVariable=13;** can be broken down into two separate statements. The statement **int myVariable;** declares the variable **myVariable**. The statement **myVariable=13;** initializes **myVariable**, giving it its initial value. See also **variable** and **variable declaration**.

write (a pin): the act of controlling an output device attached to a pin on a Circuit Playground (or other microcontroller) by sending it electrical signals. In Arduino, electrical signals are written to a pin with the statement **digitalWrite**. **digitalWrite(pin, value);** sets the pin to be either **HIGH** (maximum circuit voltage, power (+)) or **LOW** (minimum circuit voltage, 0 volts, ground(-)). See also **read** and **output** (device).

TOC

BIBLIOGRAPHY

Buechley, L., Qiu, K., Goldfein, J., & de Boer, S. (2013). *Sew Electric: A Collection of DIY Projects That Combine Fabric, Electronics, and Sewing [online version]*. Retrieved from http://sewelectric.org/.

Kitronik. (n.d.). *LED Flasher Module Kit*. Nottingham, UK: Kitronik Ltd. Retrieved from www.kitronik.co.uk/pdf/2719R_electro-fashion_flasher_module_kit_build_instructions.pdf.

Lovell, E. (n.d.) Getting hands on with soft circuits: A workshop facilitator's guide. *SparkFun*. Retrieved from https://cdn.sparkfun.com/assets/resources/2/8/guide.pdf.

Peppler, K., Gresalfi, M., Tekinbas, K. S., Santo, R., & Buechley, L. (2014). *Soft circuits: Crafting e-fashion with DIY electronics*. (pp. 61-62). Cambridge, MA: MIT Press.

Qi, J. (2014). Simple circuit tutorial. *Chibitronics, PTE LTD.* Retrieved from www.Chibitronics.com.

Strommer, A. & Fields, D. (n.d.). Light-up bracelet student guide. *Utah State University*. Retrieved from https://itls.usu.edu/files/projects/ETextiles_Bracelet_Guide.pdf.

Strommer, A. & Fields, D. (n.d.). LilyTiny Project student guide. *Utah State University*. Retrieved from https://itls.usu.edu/files/projects/ETextiles_LilyTiny_Guide.pdf.

TOC