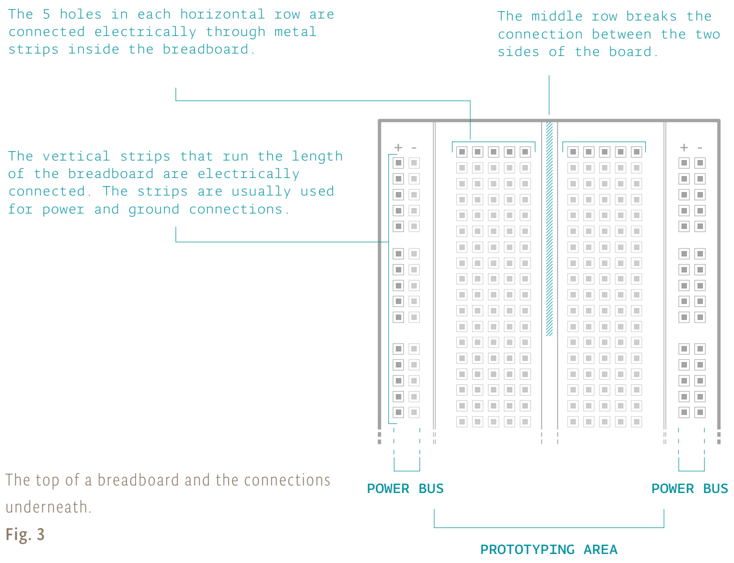
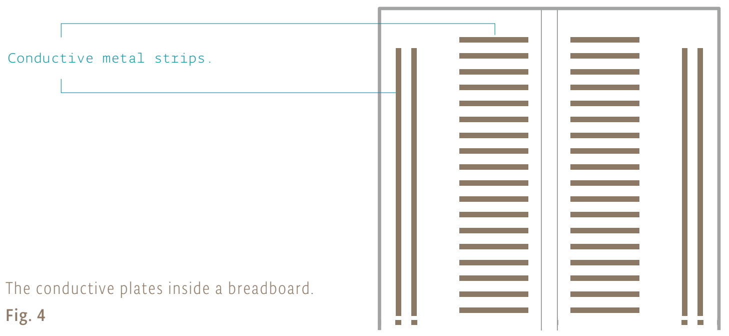
WHAT'S A BREADBOARD?

The breadboard is the primary place you will be building circuits. The one that comes in your kit is solderless, so named because you don't have to solder anything together, sort of like LEGO in electronic form. The horizontal and vertical rows of the breadboard, as shown in Fig. 3, carry electrictricity through thin metal connectors under the plastic with holes.

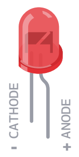

The 5 holes in each horizontal row are connected electrically through metal strips inside the breadboard.
The middle row breaks the connection between the two sides of the board.
The vertical strips that run the length of the breadboard are electrically connected. The strips are usually used for power and ground connections.
+ -
+ -
POWER BUS
POWER BUS
PROTOTYPING AREA

The top of a breadboard and the connections underneath.
Fig. 3



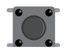
The conductive plates inside a breadboard.
Fig. 4

YOUR FIRST COMPONENTS

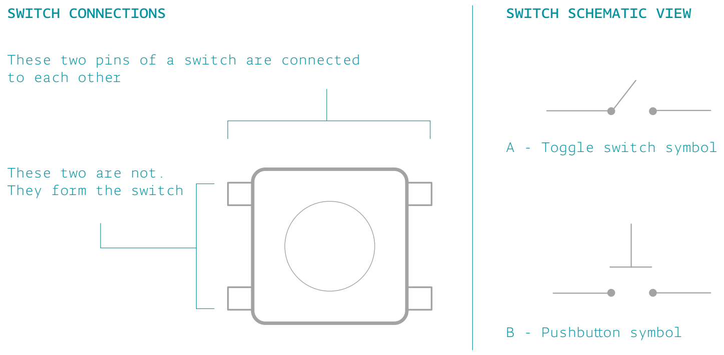


An ***LED***, or light-emitting diode, is a component that converts electrical energy into light energy. LEDs are polarized components, which means they only allow electricity to flow through them in one direction. The longer leg on the LED is called an anode, it will connect to power. The shorter leg is a cathode and will connect to ground. When voltage is applied to the anode of the LED, and the cathode is connected to ground, the LED emits light.

A ***resistor*** is a component that resists the flow of electrical energy (see the components list for an explanation on the colored stripes on the side). It converts some of the electrical energy into heat. If you put a resistor in series with a component like an LED, the resistor will use up some of the electrical energy and the LED will receive less energy as a result. This allows you to supply components with the amount of energy they need. You use a resistor in series with the LED to keep it from receiving too much voltage. Without the resistor, the LED would be brighter for a few moments, but quickly burn out.



A ***switch*** interrupts the flow of electricity, breaking the circuit when open. When a switch is closed, it will complete a circuit. There are many types of switches. The ones in your kit are called momentary switches, or pushbuttons, because they are only closed when pressure is applied.


SWITCH CONNECTIONS
These two pins of a switch are connected to each other
These two are not. They form the switch
SWITCH SCHEMATIC VIEW
A - Toggle switch symbol
B - Pushbutton symbol

The switch
Fig. 7

BUILD THE CIRCUIT

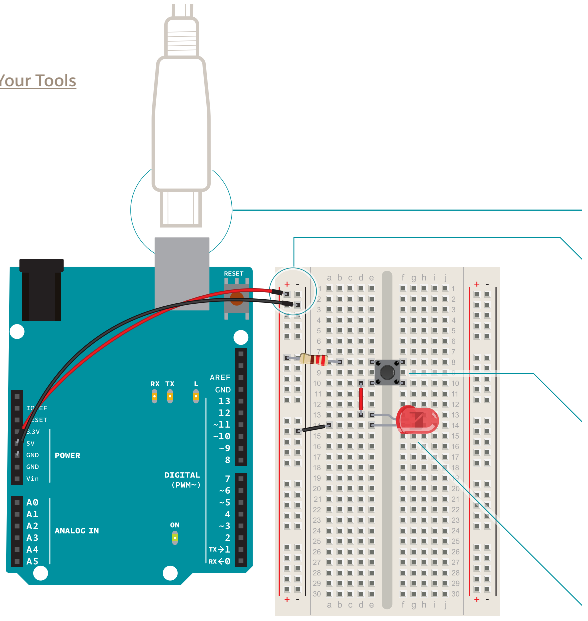


Fig. 8

Your first interactive circuit, using a switch, a resistor and an LED. Arduino is just the power source for this circuit; in later projects, you'll connect its input and output pins to control more complex circuits.

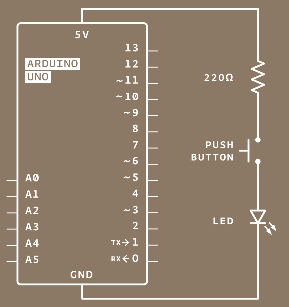


Fig. 9

You're going to use the Arduino in this project, but only as a source of power. When plugged into a USB port or a 9-volt battery, the Arduino will provide 5 volts between its 5V pin and its ground pin that you can use. 5V = 5 volts, you'll see it written this way a lot.

1. If your Arduino is connected to a battery or computer via USB, unplug it before building the circuit!

2. Connect a red wire to the 5V pin on the Arduino, and put the other end in one of the long bus lines in your breadboard. Connect ground on the Arduino to the adjacent bus line with a black wire. It's helpful to keep your wire color consistent (red for power, black for ground) throughout your circuit.

3. Now that you have power on your board, place your switch across the center of the board. The switch will sit across the center in one direction. The bend in the legs of the switch point to the center of the board.

4. Use a 220-ohm resistor to connect power to one side of the switch. The illustrations in this book use 4 bands. Your kit may have a mix of 4 and 5 band resistors. Use the illustration on the side to check for the right one for this project. Look at page 41 for a detailed explanation of the color codes for resistors.
On the other side of the switch, connect the anode (long leg) of the LED. With a wire connect the cathode (short leg) of the LED to ground. When you're ready, plug the USB cable into the Arduino.

USE IT

Once everything is set to go, press the button. You should see the LED light up. Congratulations, you just made a circuit! Once you've tired of pressing the button to turn the light on, it's time to shake things up by adding a second button.

You'll be placing components on the breadboard in series and in parallel. Components in series come one after another. Components in parallel run side by side.

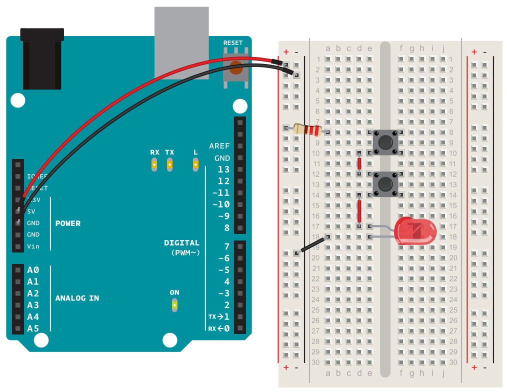
These two elements are in series

Series circuit

COMPONENTS IN SERIES COME ONE AFTER ANOTHER

Once you've removed your power source add a switch next to the one already on your breadboard. Wire them together in series as shown in Fig. 10. Connect the anode (long leg) up the LED to the second switch. Connect the LED cathode to ground. Power up the Arduino again: now to turn on the LED, you need to press both switches. Since these are in series, they both need to be closed for the circuit to be completed.

ALWAYS REMOVE POWER BEFORE CHANGING ANYTHING IN YOUR CIRCUIT



The two switches are in series. This means that the same electrical current flows through both of them, so that they both have to be pressed for the LED to light up.

Fig. 10

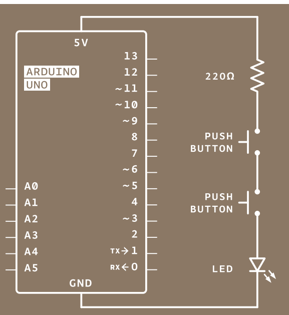
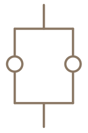


Fig. 11

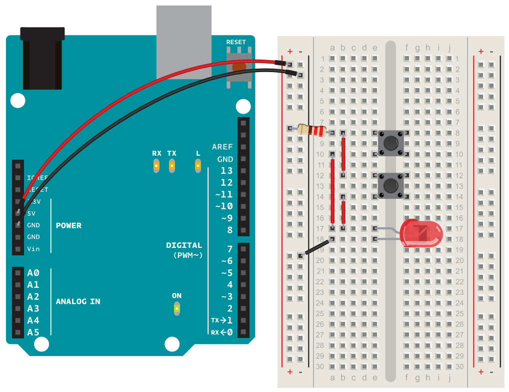


These two elements are in parallel

Parallel circuit

COMPONENTS IN PARALLEL RUN SIDE BY SIDE

Now that you've mastered the art of things in series, it's time to wire up switches in parallel. Keep the switches and LED where they are, but remove the connection between the two switches. Wire both switches to the resistor. Attach the other end of both switches to the LED, as shown in Fig. 12. Now when you press either button, the circuit is completed and the light turns on.



These two switches are in parallel. This means that the electrical current is split between them. If either switch is pressed, the LED will light up.

Fig. 12

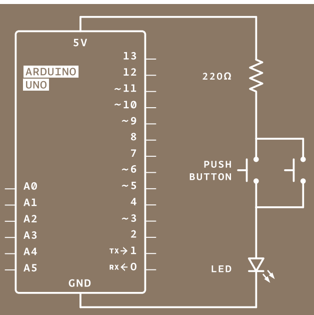
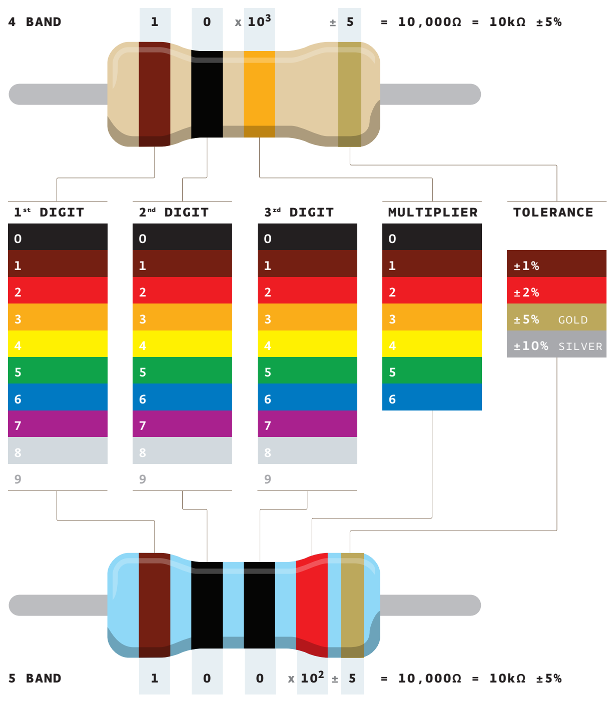


Fig. 13

HOW TO READ RESISTOR COLOR CODES

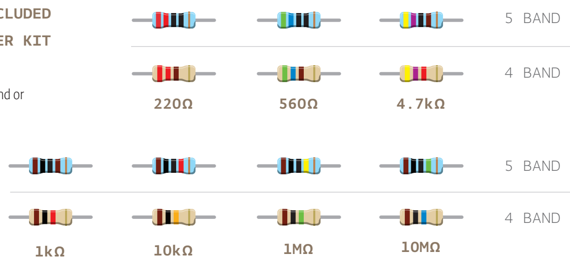
Resistor values are marked using colored bands, according to a code developed in the 1920s, when it was too difficult to write numbers on such tiny objects.
Each color corresponds to a number, like you see in the table below. Each resistor has either 4 or 5 bands. In the 4-band type, the first two bands indicate the first two digits of the value while the third one indicates the number of zeroes that follow (technically it reprents the power of ten). The last band specifies the tolerance: in the example below, gold indicates that the resistor value can be 10k ohm plus or minus 5%.



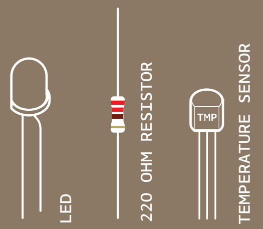
1st DIGIT	2nd DIGIT	3rd DIGIT	MULTIPLIER	TOLERANCE
⬛ 0	⬛ 0	⬛ 0	⬛ 0	
🟫 1	🟫 1	🟫 1	🟫 1	🟫 ±1%
🟥 2	🟥 2	🟥 2	🟥 2	🟥 ±2%
🟧 3	🟧 3	🟧 3	🟧 3	±5% GOLD
🟨 4	🟨 4	🟨 4	🟨 4	±10% SILVER
🟩 5	🟩 5	🟩 5	🟩 5	
🟦 6	🟦 6	🟦 6	🟦 6	
🟪 7	🟪 7	🟪 7		
▨ 8	▨ 8	▨ 8		
⬜ 9	⬜ 9	⬜ 9		

RESISTORS INCLUDED IN THE STARTER KIT

You'll find either a 4 band or a 5 band version.



03



INGREDIENTS

BUILD THE CIRCUIT

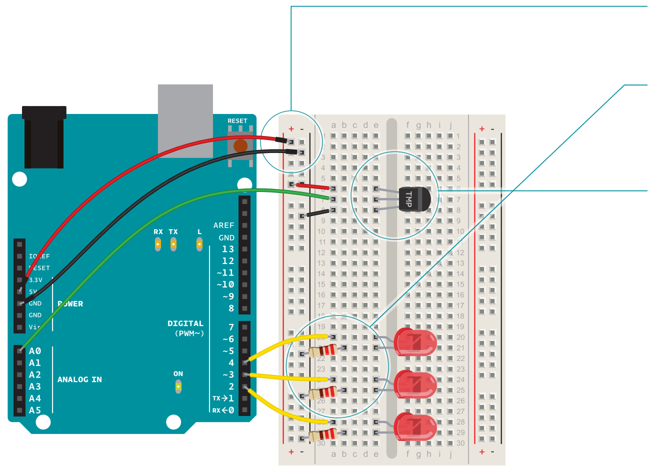


Fig. 2

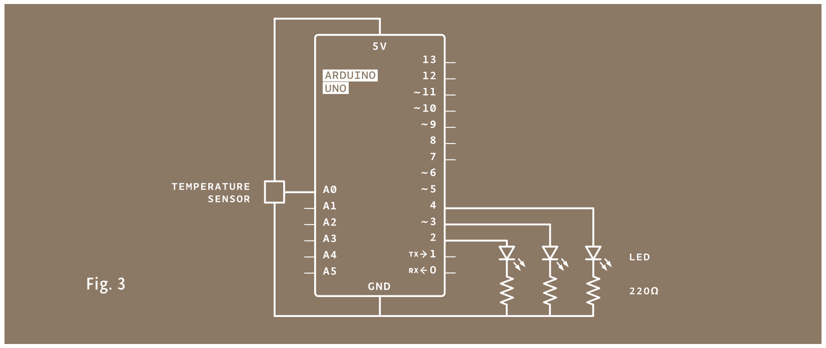


Fig. 3

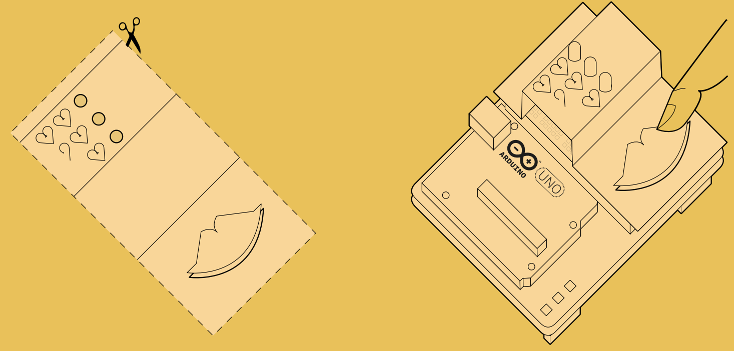
In this project, you need to check the ambient temperature of the room before proceeding. You're checking things manually right now, but this can also be accomplished through calibration. It's possible to use a button to set the baseline temperature, or to have the Arduino take a sample before starting the `loop()` and use that as the reference point. Project 6 gets into details about this, or you can look at the Calibration example that comes bundled with the Arduino software: ***arduino.cc/calibration***

1. Just as you've been doing in the earlier projects, wire up your breadboard so you have power and ground.

2. Attach the cathode (short leg) of each of the LEDs you're using to ground through a 220-ohm resistor. Connect the anodes of the LEDs to pins 2 through 4. These will be the indicators for the project.

3. Place the TMP36 on the breadboard with the rounded part facing away from the Arduino (the order of the pins is important!) as shown in Fig. 2. Connect the left pin of the flat facing side to power, and the right pin to ground. Connect the center pin to pin A0 on your Arduino. This is analog input pin 0.

Create an interface for your sensor for people interact with. A paper cutout in the shape of a hand is a good indicator. If you're feeling lucky, create a set of lips for someone to kiss, see how well that lights things up! You might also want to label the LEDs to give them some meaning. Maybe one LED means you're a cold fish, two LEDs means you're warm and friendly, and three LEDs means you're too hot to handle!



1. Cut out a piece of paper that will fit over the breadboard. Draw a set of lips where the sensor will be, and cut some circles for the LEDs to pass through.

2. Place the cutout over the breadboard so that the lips cover the sensor and the LEDs fit into the holes. Press the lips to see how hot you are!

THE CODE

A pair of useful constants

Constants are similar to variables in that they allow you to uniquely name things in the program, but unlike variables they cannot change. Name the analog input for easy reference, and create another named constant to hold the baseline temperature. For every 2 degrees above this baseline, an LED will turn on.
You've already seen the int datatype, used here to identify which pin the sensor is on. The temperature is being stored as a *float*, or floating-point number. This type of number has a decimal point, and is used for numbers that can be expressed as fractions.

```
const int sensorPin = A0;
const float baselineTemp = 20.0;
```

Initialize the serial port to the desired speed

In the setup you're going to use a new command, `Serial.begin()`. This opens up a connection between the Arduino and the computer, so you can see the values from the analog input on your computer screen.
The argument `9600` is the speed at which the Arduino will communicate, 9600 bits per second. You will use the Arduino IDE's serial monitor to view the information you choose to send from your microcontroller. When you open the IDE's serial monitor verify that the baud rate is 9600.

```
void setup(){
  Serial.begin(9600); // open a serial port
```

Initialize the digital pin directions and turn off

Next up is a `for()` loop to set some pins as outputs. These are the pins that you attached LEDs to earlier. Instead of giving them unique names and typing out the `pinMode()` function for each one, you can use a `for()` loop to go through them all quickly. This is a handy trick if you have a large number of similar things you wish to iterate through in a program. Tell the `for()` loop to run through pins 2 to 4 sequentially.

```
  for(int pinNumber = 2; pinNumber<5; pinNumber++){
    pinMode(pinNumber,OUTPUT);
    digitalWrite(pinNumber, LOW);
  }
}
```

`for() loop tutorial`
arduino.cc/for

Read the temperature sensor

In the `loop()`, you'll use a local variable named `sensorVal` to store the reading from your sensor. To get the value from the sensor, you call `analogRead()` that takes one argument: what pin it should take a voltage reading on. The value, which is between 0 and 1023, is a representation of the voltage on the pin.

```
void loop(){
  int sensorVal = analogRead(sensorPin);
```

Send the temperature sensor values to the computer

The function `Serial.print()` sends information from the Arduino to a connected computer. You can see this information in your serial monitor. If you give `Serial.print()` an argument in quotation marks, it will print out the text you typed. If you give it a variable as an argument, it will print out the value of that variable.

```
  Serial.print("Sensor Value: ");
  Serial.print(sensorVal);
```

Convert sensor reading to voltage

With a little math, it's possible to figure out what the real voltage on the pin is. The voltage will be a value between 0 and 5 volts, and it will have a fractional part (for example, it might be 2.5 volts), so you'll need to store it inside a `float`. Create a variable named voltage to hold this number. Divide `sensorVal` by 1024.0 and multiply by 5.0. The new number represents the voltage on the pin.

```
// convert the ADC reading to voltage
float voltage = (sensorVal/1024.0) * 5.0;
```

Just like with the sensor value, you'll print this out to the serial monitor.

```
Serial.print(", Volts: ");
Serial.print(voltage);
```

Convert the voltage to temperature and send the value to the computer

If you examine the sensor's *datasheet*, there is information about the range of the output voltage. Datasheets are like manuals for electronic components. They are written by engineers, for other engineers. The datasheet for this sensor explains that every 10 millivolts of change from the sensor is equivalent to a temperature change of 1 degree Celsius. It also indicates that the sensor can read temperatures below 0 degrees. Because of this, you'll need to create an offset for values below freezing (0 degrees). If you take the voltage, subtract 0.5, and multiply by 100, you get the accurate temperature in degrees Celsius. Store this new number in a floating point variable called temperature.

Now that you have the real temperature, print that out to the serial monitor too. Since the temperature variable is the last thing you're going to be printing out in this loop, you're going to use a slightly different command: `Serial.println()`. This command will create a new line in the serial monitor after it sends the value. This helps make things easier to read in when they are being printed out.

```
Serial.print(", degrees C: ");
// convert the voltage to temperature in degrees
float temperature = (voltage - .5) * 100;
Serial.println(temperature);
```

Starter Kit datasheets
arduino.cc/kitdatasheets

Turn off LEDs for a low temperature

With the real temperature, you can set up an `if()...else` statement to light the LEDs. Using the baseline temperature as a starting point, you'll turn on one LED on for every 2 degrees of temperature increase above that baseline. You're going to be looking for a range of values as you move through the temperature scale.

```
if(temperature < baselineTemp){
  digitalWrite(2, LOW);
  digitalWrite(3, LOW);
  digitalWrite(4, LOW);
```

Turn on one LED for a low temperature

The && operator means "and", in a logical sense. You can check for multiple conditions: "if the temperature is 2 degrees greater than the baseline, and it is less than 4 degrees above the baseline."

```
  }else if(temperature >= baselineTemp+2 &&
    temperature < baselineTemp+4){
    digitalWrite(2, HIGH);
    digitalWrite(3, LOW);
    digitalWrite(4, LOW);
```

Turn on two LEDs for a medium temperature

If the temperature is between two and four degrees above the baseline, this block of code turns on the LED on pin 3 as well.

```
  }else if(temperature >= baselineTemp+4 &&
    temperature < baselineTemp+6){
    digitalWrite(2, HIGH);
    digitalWrite(3, HIGH);
    digitalWrite(4, LOW);
```

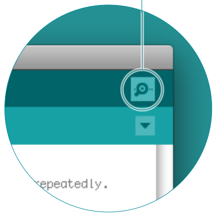
Turn on three LEDs for a high temperature

```
  }else if(temperature >= baselineTemp+6){
    digitalWrite(2, HIGH);
    digitalWrite(3, HIGH);
    digitalWrite(4, HIGH);
```

The Analog-to-Digital Converter can only read so fast, so you should put a small delay at the very end of your `loop()`. If you read from it too frequently, your values will appear erratic.

```
  }
  delay(1);
}
```

USE IT



With the code uploaded to the Arduino, click the serial monitor icon. You should see a stream of values coming out, formatted like this : `Sensor: 200, Volts: .70, degrees C: 17`

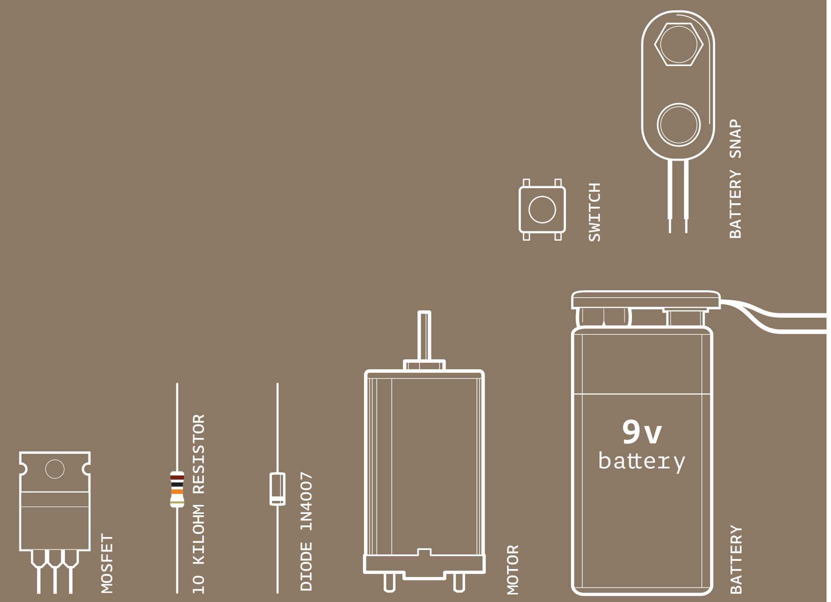
Try putting your fingers around the sensor while it is plugged into the breadboard and see what happens to the values in the serial monitor. Make a note of what the temperature is when the sensor is left in the open air.

Close the serial monitor and change the baselineTemp constant in your program to the value you observed the temperature to be. Upload your code again, and try holding the sensor in your fingers. As the temperature rises, you should see the LEDs turn on one by one. Congratulations, hot stuff!

Create an interface for two people to test their compatibility with each other. You get to decide what compatibility means, and how you'll sense it. Perhaps they have to hold hands and generate heat? Maybe they have to hug? What do you think?

Expanding the types of inputs you can read, you've used analogRead() and the serial monitor to track changes inside your Arduino. Now it's possible to read a large number of analog sensors and inputs.

09



INGREDIENTS

MOTORIZED PINWHEEL

GET THE ARDUINO TO SPIN A COLORFUL PINWHEEL USING A MOTOR

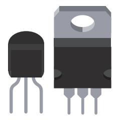
Discover: transistors, high current/voltage loads

Time: **45 MINUTES**
Level: ■■■□□

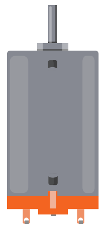
Builds on projects: **1, 2, 3, 4**

Controlling motors with an Arduino is more complicated than just controlling LEDs for a couple of reasons. First, motors require more current than the Arduino's output pins can supply, and second, motors can generate their own current through a process called induction, which can damage your circuit if you don't plan for it. However, motors make it possible to move physical things, making your projects much more exciting. They're worth the complications!

Moving things takes a lot of energy. Motors typically require more current than the Arduino can provide. Some motors require a higher voltage as well. To start moving, and when it has a heavy load attached, a motor will draw as much current as it can. The Arduino can only provide 40 milliamps (mA) from its digital pins, much less than what most motors require to work.



Transistors are components that allow you to control high current and high voltage power sources from the low current output of the Arduino. There are many different kinds, but they work on the same principle. You can think of transistors as digital switches. When you provide voltage to one of the transistor's pins, called the gate, it closes the circuit between the other two pins, called the source and drain. This way, you can turn a higher current/voltage motor on and off with your Arduino.



Motors are a type of inductive device. Induction is a process by which a changing electrical current in a wire can generate a changing magnetic field around the wire. When a motor is given electricity, a tightly wound coil inside the housing of copper creates a magnetic field. This field causes the shaft (the part that sticks out of the housing) to spin around.

The reverse is also true: a motor can generate electricity when the shaft is spun around. Try attaching an LED to the two leads of your motor, then spin the shaft with your hand. If nothing happens, spin the shaft the other way. The LED should light up. You've just made a tiny generator out of your motor.
When you stop supplying energy to a motor, it will continue to spin, because it has inertia. When it's spinning, it will generate a voltage in the opposite direction than the current you gave it. You saw this effect when you made your motor light up an LED. This reverse voltage, sometimes called back-voltage, can damage your transistor. For this reason, you should put a diode in parallel with the motor, so that the *back voltage* passes through the diode. The diode will only allow electricity to flow in one direction, protecting the rest of the circuit.

BUILD THE CIRCUIT

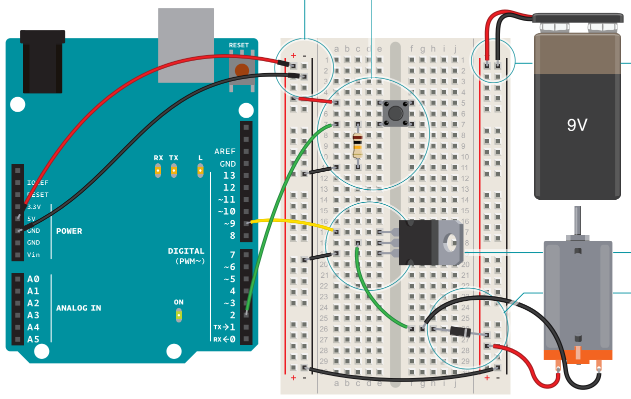


Fig. 1

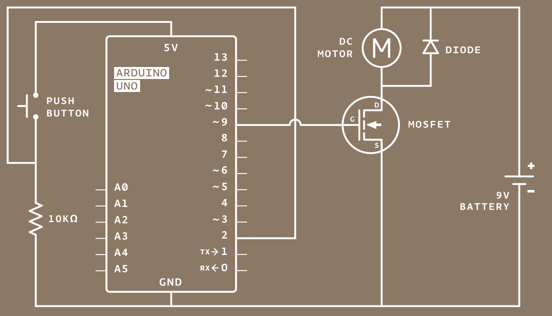


Fig. 2

1. Connect power and ground to your breadboard through the Arduino.

2. Add a momentary switch to the board, connecting one side to power, and the other side to digital pin 2 on the Arduino. Add a 10-kilohm pull-down resistor to ground on the output pin of the switch.

3. When using circuits with different voltages, you have to connect their grounds together to provide a common ground. Plug the 9V battery snap into your breadboard. Connect ground from the battery to ground of your Arduino on the breadboard with a jumper, as shown in Fig. 1. Then attach the motor's free lead to the 9V power.

4. Place the transistor on the board. Look at the component so that the metal tab is facing away from you. Connect digital pin 9 to the left pin on the transistor. This pin is called the **gate**. A change in voltage on the gate makes a connection between the other two pins. Connect one end of the motor to the middle pin of the transistor. This pin is called the **drain**. When the Arduino activates the transistor by supplying voltage to the gate, this pin will be connected to the third pin, called the **source**. Connect the source to ground.

5. Next, connect the motor's voltage supply to the motor and breadboard. The last component to be added is the diode. The diode is a polarized component, it can go only one way in the circuit. Notice that the diode has a stripe on one end. That end is the negative end, or cathode, of the diode. The other end is the positive end, or anode. Connect the anode of the diode to the ground of the motor and the cathode of the diode to the power of the motor. See Fig. 1. This may seem backwards, and in fact, it is. The diode will help prevent any back-voltage generated by the motor from going back into your circuit. Remember, back voltage will flow in the opposite direction of the voltage that you supply.

LEDs are diodes too, in case you were wondering why their leads were also called anodes and cathodes. There are many kinds of diodes, but they all share one trait. They allow current to flow from anode to cathode, but not the reverse.

THE CODE

Name your constants and variables

The code is remarkably similar to the code you first used for turning on an LED. First of all, set up some constants for the switch and motor pins and a variable named `switchState` to hold the value of the switch.

```
const int switchPin = 2;
const int motorPin = 9;
int switchState = 0;
```

Declare the pins' direction

In your `setup`(), declare the `pinMode`() of the motor (OUTPUT) and `switch` (INPUT) pins.

```
void setup() {
  pinMode(motorPin, OUTPUT);
  pinMode(switchPin, INPUT);
}
```

Read the input, pull the output high if pressed

Your `loop`() is straightforward. Check the state of the `switchPin` with `digitalRead`().

If the switch is pressed, turn the motorPin HIGH. If it is not pressed, turn the pin LOW. When HIGH, the transistor will activate, completing the motor circuit. When LOW, the motor will not spin.

```
void loop(){
  switchState = digitalRead(switchPin);

  if (switchState == HIGH) {
    digitalWrite(motorPin, HIGH);
  }
  else {
    digitalWrite(motorPin, LOW);
  }
}
```

Motors have an optimal operating voltage. They will work on as little as 50% of the rated voltage and as much as 50% over that number. If you vary the voltage, you can change the speed at which the motor rotates. Don't vary it too much, though, or you will burn out your motor.

Motors require special consideration when being controlled by a microcontroller. Typically the microcontroller cannot provide enough current and/or voltage to power a motor. Because of this, you use transistors to interface between the two. It's also smart to use diodes to prevent damaging your circuit.

Transistors are solid state devices, they have no moving parts. Because of this, you can switch them on and off very quickly. Try hooking up a potentiometer to an analog input and use that to PWM the pin that controls the transistor. What do you think will happen to the motor's speed if you vary the voltage it's getting? Using your patterns on your spinner, can you get different visual effects?