Raspberry Pi LED Birthday Candle

Tutorial by Andrew Oakley

www.cotswoldjam.org

Introduction

This tutorial shows you how to build a birthday candle based around an LED (light) and a photocell (light detector) using a Raspberry Pi computer.

You'll be able to "blow out" the LED - and it'll turn off and play a tune!

Getting started

Conventions

```
Words you will see on the screen are highlighted like this
Words you need to type in are highlighted and underlined
```

At the end of each line, you will usually have to press the Enter key.

Your tutor should have prepared your Raspberry Pi for you. If not, see the section “Preparation” at the end.

The circuit

Build up the circuit as follows. Place the components first, and the jumper wires last. Don't worry about the paperclips just yet.

The LED has a short leg and a long leg. If you feel around the rim, you'll also find a flat edge. The short leg and flat edge always connect to negative (ground).

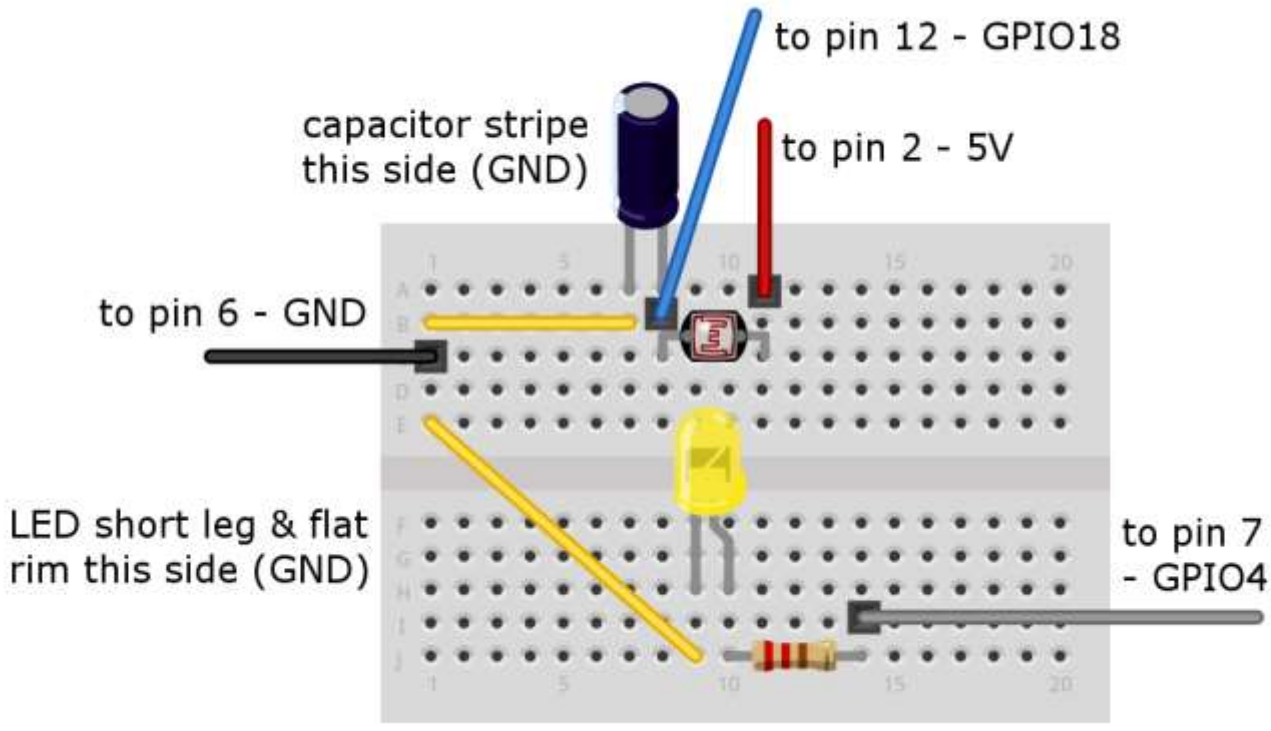
LEDs always need to be connected with a resistor. If you connect them without a resistor, they will burn out and probably won't work again.

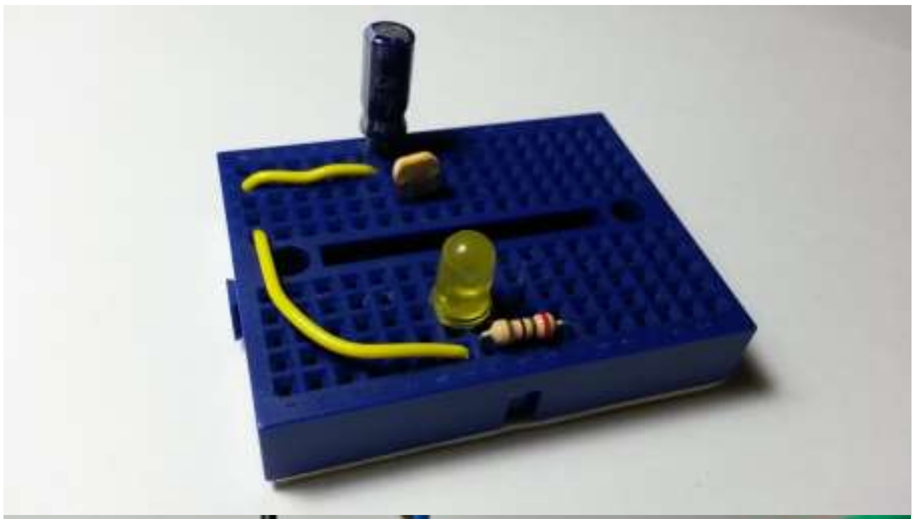
The resistor can be connected any way around. Lower value (lower ohm) resistors will allow more current through and will make the LED brighter; higher values will let less current through and make it dimmer. We're using a 270 ohm resistor but anything between 220-470 ohms will work fine.

The photocell, or photoresistor, can be connected any way around. They let through more current when there is more light. We're using one that is rated between 2 and 20,000 ohms but most photocells you find in hobby kits will work fine.

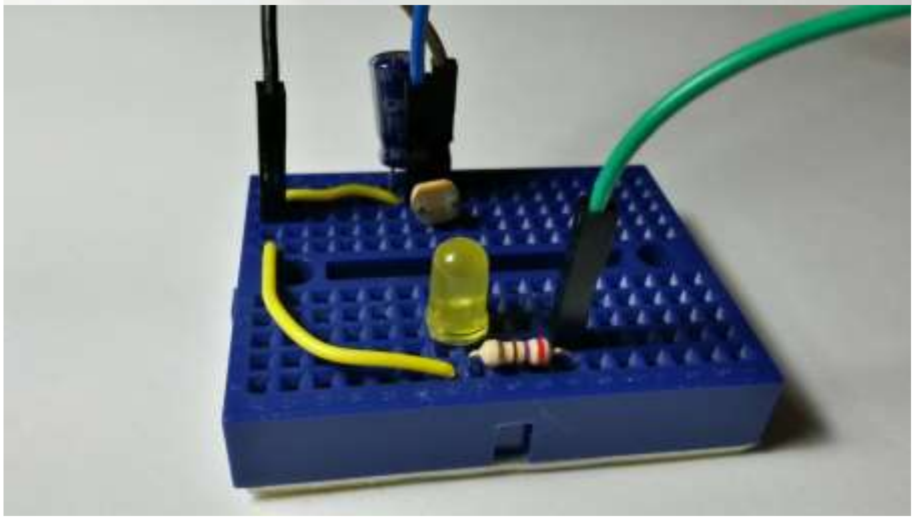
The capacitor has a short and a long leg, and a stripe on one side. Like LEDs, they must be connected with the short leg and stripe to negative (ground). We're using a 1 microfarad capacitor and you should too.

A capacitor stores a small amount of charge, then when it is full it lets all the charge go. If you connect it to a stronge charge, it lets the charge go faster. If you connect it to a weak charge, it takes longer to let the charge go.


to pin 12 - GPIO18
capacitor stripe this side (GND)
to pin 2 - 5V
to pin 6 - GND
LED short leg & flat rim this side (GND)
to pin 7 - GPIO4



Here's what the breadboard should look like before you plug in the jumper wires.

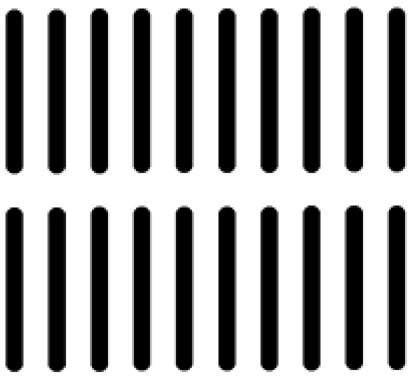


And here's the breadboard with the jumper wires attached.

Important - bend the photocell to face the light

You will need to bend the top of the photocell so that it is "looking" towards the LED. See the photos for an example.

About breadboards



Breadboards are used to build and test electronic circuits without having to solder the components in place. They are reusable, and the components are also reusable after being used on a breadboard.

All the vertical lines (usually lettered A-E and F-J) are connected in columns. So electricity going into point 1A also arrives at 1B, 1C, 1D and 1E. But there are no connections across rows - so electricity going into point 1A does not arrive at 2A, 3A and so on.

Some breadboards also have extra rows at the top or bottom which are connected vertically, used for positive and negative. We are not using that type of breadboard, and if you are using one then ignore the rows at the top and bottom of your board. Sometimes these rows are marked with plus + and minus - or are coloured red and blue.

Leg trimming

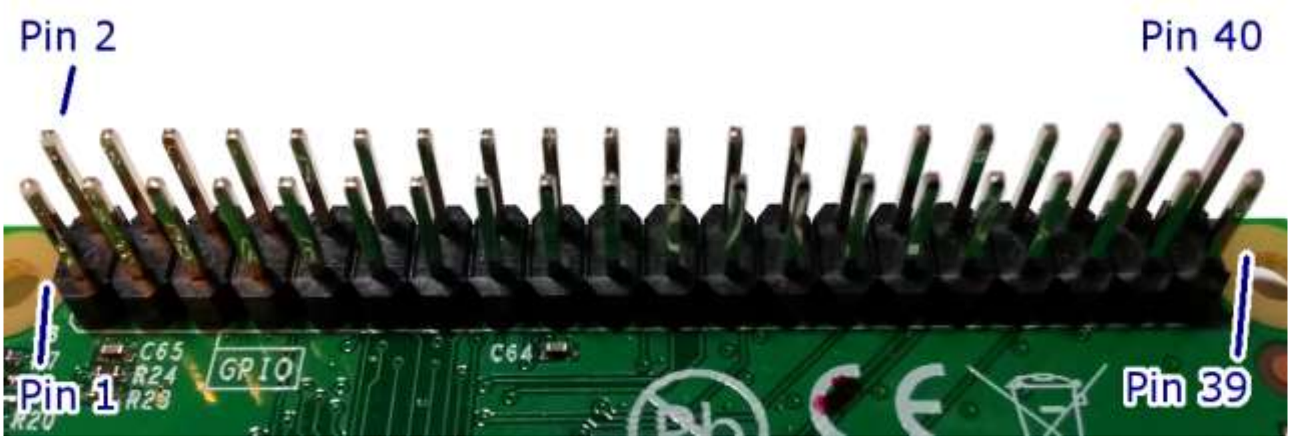
When components come out of the factory, they have long legs (wires). **Your tutor may have already trimmed the legs of your components, in which case you do not need to do it again.** If your components have long legs, then before you cut them, you should try out your circuit by connecting it to the Raspberry Pi and running the programs.

If and when you need to trim the legs, you can use kitchen scissors or a special wire snipping tool. Either way, be careful when cutting the legs of components:

- **Metal will fly off when cutting and can be very dangerous near your eyes** - wear goggles or place the component under a tea-towel when snipping the legs.
- Don't trim the legs so short that they don't stick into the breadboard holes.
- For LEDs and capacitors, remember you can use the LED rim or capacitor stripe to tell positive from negative, even if you trim both legs the same length.

Connecting the Raspberry Pi

Connect the jumper wires as per the diagram. The pins on the Raspberry Pi's GPIO port are numbered as follows. On older models there will only be 26 pins, but they still start from 1 and 2 on the left side. The top row has evens, the bottom row has odd numbers.



The test programs

Power up your Raspberry Pi, log in and go to the desktop. If you need to log in, the default username is `pi` and the password is `raspberry` . If the desktop still does not appear, type `startx` and press the enter key.

From the menu, select Programming - Python 2 (not Python 3). Then use File, New Window to create a new program. Click File, Save As and go into the python/candle folder by double-clicking python and then double-clicking candle. Save as: `mycandle.py`

Lighting up the LED

Let's start by lighting up the LED. Type in the following program, or load it from mycandle1.py in the examples folder:

```
import RPi.GPIO as GPIO, time

GPIO.setmode(GPIO.BOARD)
GPIO.setwarnings(False)
pin_write=7
GPIO.setup(pin_write, GPIO.OUT)

GPIO.output(pin_write, GPIO.HIGH)
time.sleep(2)
GPIO.output(pin_write, GPIO.LOW)
```

Save the program using File - Save. Now leave the Python window open, and start a terminal session by going to the main desktop Menu - Accessories - Terminal. You should see a black window open.

In the terminal, type:

```
cd python/candle
sudo python mycandle.py
```

You should see the LED light up for two seconds, and then the program will end. You can repeat the program by entering **`sudo python mycandle.py`** again.

Let's have a quick look at what this program is doing.

The **`import`** command tells the computer to use bits of another program that have been written by someone else. These other programs are called libraries. We call in two libraries; **`GPIO`** which lets us use the GPIO pins, and **`time`** which includes the sleep command which we use to wait for two seconds.

Next we use the **`GPIO.setmode(GPIO.BOARD)`** command to tell the computer that we're going to count the GPIO pins using the numbers 1-40 (or 1-26 if you have an older Raspberry Pi). There is another option to use "BCM" rather than board, which counts the pins in a different way. **`GPIO.setwarnings(False)`** stops the computer complaining if we have set the mode already - for example, if we run the program more than once.

`pin_write=7` is a variable. Variables can hold all kinds of values such as numbers or letters. We then **`setup`** pin_write (which is pin 7) to be an output - it is going to send electricity.

At last we're onto the command which actually lights up the LED! The command **`GPIO.output(pin_write, GPIO.HIGH)`** turns on the LED. We then wait two seconds with **`time.sleep(2)`** before we turn off the LED with **`GPIO.output(pin_write, GPIO.LOW`**)

If it didn't work, check your spelling and your breadboard connections.

Sensing light

We used a GPIO output to send electricity to the LED. Now let's use a GPIO input to read how much light our photocell can detect.

The problem here is that the Raspberry Pi's GPIO pins are digital - they can only detect ones and zeroes; high voltage and low voltage. Unfortunately a photocell gives a range of voltages depending on how bright the light is - you can never really be sure when the brightness crosses over from a zero to a one.

This is where the capacitor comes in. The capacitor stores a tiny amount of charge and when it gets full, it lets it all go in one big lump - which the Raspberry Pi can detect as high voltage.

When it's dark, the photocell will have high resistance and will charge the capacitor slowly.

When it's bright, the photocell will have low resistance and will charge the capacitor quickly.

So we will use the Raspberry Pi to count how long it takes to charge the capacitor. A short count means a bright light; a long count means dark.

Change your program to the following, or load it from mycandle2.py in the examples folder:

```
import RPi.GPIO as GPIO, time

GPIO.setmode(GPIO.BOARD)
GPIO.setwarnings(False)
pin_write=7
pin_read=12

GPIO.setup(pin_write, GPIO.OUT)
GPIO.output(pin_write, GPIO.HIGH)
time.sleep(1)

GPIO.setup(pin_read, GPIO.OUT)
GPIO.output(pin_read, GPIO.LOW)
time.sleep(0.01)
GPIO.setup(pin_read, GPIO.IN)

reading=0
while (GPIO.input(pin_read) == GPIO.LOW):
  reading += 1

print "Reading: %d" % reading

GPIO.output(pin_write, GPIO.LOW)
```

Run the program just like last time, by switching to the terminal window and entering **`sudo python mycandle.py`** .

If the program gets stuck, hold down the CTRL key and press C. Then check your wiring and the spelling in your program.

If the program works, it will light the LED for a second, then start counting. It then says what the count was. A small count means light, a big count means dark.

Try running the program with your finger blocking the photocell. Try using bits of black card to put the photocell into shadow. If you're trying this in a well-lit room, the room lights might be more powerful than the LED - later we will make a blackout hood using paperclips and card.

In the program, you can see that we briefly (for 0.1 seconds) set pin_read to an output and set the voltage low, just to clear any voltage on the connection, before we set it to an input with **`GPIO.setup(pin_read, GPIO.IN)`** .

We then start counting. We have a new variable, **`reading`** , which we set to zero.

The program then loops around until it detects high voltage from the capacitor, adding one to the **`reading`** count every time. When it detects high voltage, it ends the loop and prints out the value of the **`reading`** .

Hoods and flaps - detecting breath

We are going to use a little flap of card to detect breath. We're going to make a U-shaped hoop out of a paperclip, then hang a bit of card from it.

We'll place that hoop and card between the LED and the photocell, and make sure it can get blown out of the way when we blow on it.

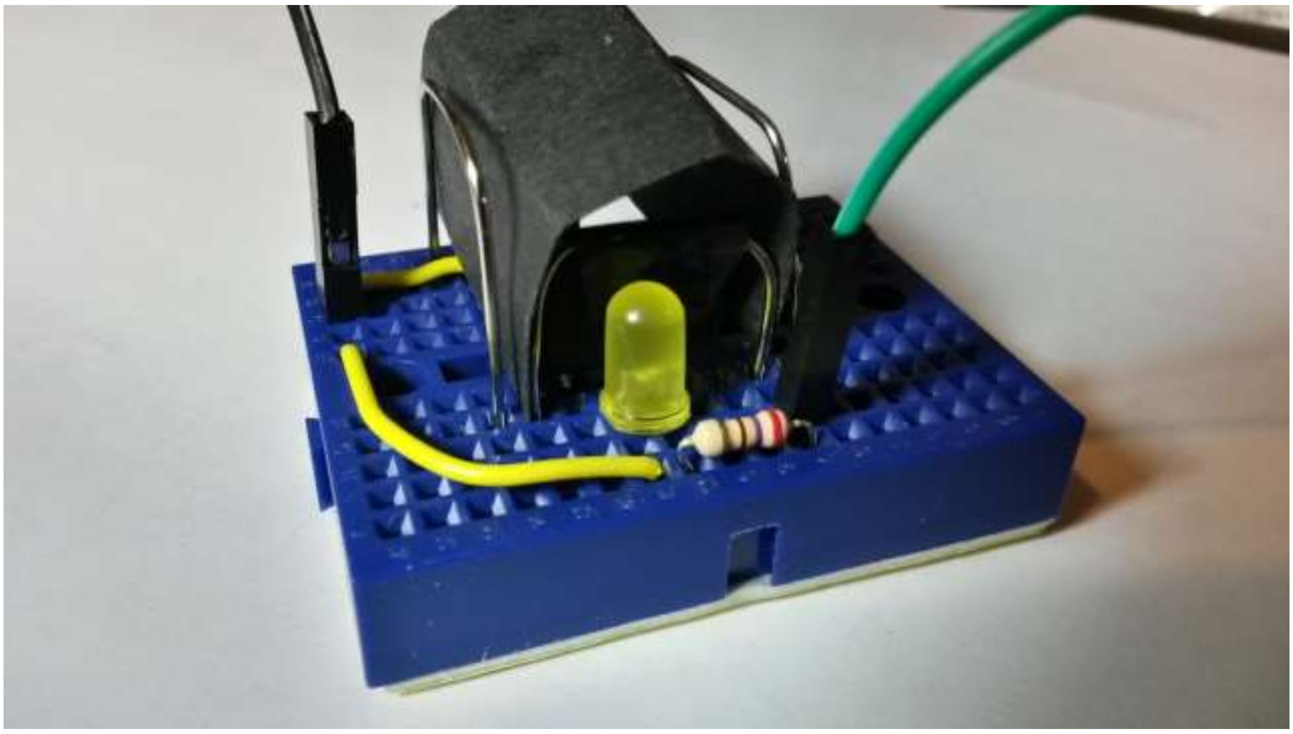
We'll also put a hood over the photocell so that the only light it can see, will be from the LED. We'll use another two larger paperclip U-hoops to hold the card in place. Paperclips are quite handy with electronics breadboards as they fit firmly in the holes!

Have a good look at the following photo, and create your own hood and flap.

Your tutor may have already bent and cut your paperclips to size. If not, be careful of flying metal when you snip your paperclips to the right length.

The flap is made from:

- A piece of black card approx. 2cm x 1cm
- A paperclip hoop approx. 1.5cm tall and 1.5cm wide
- Bend one end of the card gently (bend don't fold) over the hoop and use a very small amount of sticky tape to hold it loosely. Remember it has to be loose and delicate enough to swing when you blow it. It you tape it down too tight, it won't swing and it won't work.



The hood is made from:

- A piece of black card approx. 4.5cm x 2cm
- Two paperclip hoops approx. 2.5cm tall x 1.5cm across
- Just gently bend the card and hold it in place with the hoops

The final program

The final program is based on the second test program:

- It sets up the GPIO pins and lights the LED
- It does some "calibration" - it works out what the reading for "dark" is, when the flap is closed
- It loops around waiting for a new reading which is significantly lower than the calibration reading. This lower reading should mean there is much more light on the photocell - and that therefore the flap has been blown out of the way!
- There is a variable **`sensitivity`** which controls how much lower the reading has to be for the program to detect the flap has been blown. By default it is set to 50% but you can change this. Remember you can use CTRL-C to stop your program if it gets stuck.
- When the program detects light, it plays the tune "Good Morning To All" from 1893. This tune is out of copyright, but you may decide to sing along with some birthday-related lyrics at your own risk.

This program is rather long to type in, so you can load it as **`candle.py`** from the **`examples`** folder if you prefer.

```
#!/usr/bin/env python

# Raspberry Pi GPIO Birthday Candles by Andrew Oakley aoakley.com
Public Domain 2015
# Adapted from "Basic Photocell Reading" by Adafruit

# Import the libraries we will be using
import RPi.GPIO as GPIO, time, os
# RPi.GPIO - for controlling GPIO pins
# time - for waiting
# os - for calling the music program

# Set up the GPIO
GPIO.setmode(GPIO.BOARD)
GPIO.setwarnings(False)
pin_read=12 # GPIO 18
pin_write=7 # GPIO 4

# Sensitivity, in percent
sensitivity=50

def timeLowHigh ():
  # This times how long it takes for a pin to go from low to high.
  # It is handy for reading variable resistance using a capacitor.
  # If the capacitor fills slowly, there is high resistance.
  # Photocells are a type of variable resistor. When there is low
  # light, then there is high resistance.
  # Therefore long time = darker, short time = lighter
  # Try using a 1 microfarad capacitor and a 2-20K ohm photocell.

  # Reset the reading to zero
  reading = 0

  # Reset the pin to low - we briefly set it to output
  GPIO.setup(pin_read, GPIO.OUT)
  GPIO.output(pin_read, GPIO.LOW)
  # Brief sleep just to let it settle on low
  time.sleep(0.01)

  # Now change the pin back to an input
  GPIO.setup(pin_read, GPIO.IN)
  # This takes about 1 millisecond per loop cycle
  while (GPIO.input(pin_read) == GPIO.LOW):
    reading += 1

  return reading
```

```
def calibrate():
  # This turns on the LED and works out what the
  # "normal" level of dimness is with the LED on.

  # Turn on the LED
  GPIO.setup(pin_write, GPIO.OUT)
  GPIO.output(pin_write, GPIO.HIGH)

  # Read the dimness lots of times over 3 seconds
  # and keep a running tally of the average
  print ("Calibrating...")
  dimness=timeLowHigh()
  t=time.time()
  while time.time() < t+2:
    c=timeLowHigh()
    print "   calibration: %05d  now: %05d" % (dimness, c)
    dimness=( dimness+c )/2

  return dimness

# Record what normal dimness is
normal_dimness=calibrate()

while True:
  # Find out what the dimness is now
  dimness_now=timeLowHigh()
  # Print out the values
  print("dimness normal: %05d  now: %05d  target: %05d" %
(normal_dimness, dimness_now, normal_dimness*(sensitivity/100.0) )
)

  # Has the dimness fallen below the target?
  if dimness_now < normal_dimness*(sensitivity/100.0) :
    GPIO.output(pin_write, GPIO.LOW)
    print ("Good Morning to All!")
    os.system('omxplayer good-morning-to-all.ogg')
    normal_dimness=calibrate()
```

As before, run the program by switching to the terminal window and entering **`sudo python candle.py`** or **`mycandle.py`** depending on how you named it.

The program will run in a loop. To stop it, hold down the CTRL key and press C.

Preparation

If your tutor has not prepared your Raspberry Pi and equipment, or if you don't have a tutor, you can set things up by following these instructions before you start.

Equipment

You will need:

- A Raspberry Pi running the Raspbian operating system. This tutorial suits all models to date, which includes the A, A+, B, B+ and 2B.
- A solderless breadboard – we're using a mini 170-point one, but any solderless breadboard will do.
- An LED. We're using a 5mm yellow one. It needs to be bright enough to trigger the photocell, so yellow, green or white will usually work beter than red or blue.
- A photocell, also known as a photoresistor. We're using a 4mm 2-20k ohm one.
- Four female-to-female jumper wires, at least 20cm long each.
- Two short male-to-male jumper wires, about 2-6cm each. We're using solid wire.
- Three paperclips, cut into U-shapes.
- Two small bits of black card; one about 4.5 x 2cm, and one about 2 x 1cm

Files

Please use an up-to-date installation of Raspbian. This tutorial was written in April 2015 and, at the time, the most recent recommended Raspbian image was dated 2015-02-16.

You can create the folder structure and download the files by going to the terminal (Menu - Accessories - Terminal) and typing:

```
cd
mkdir python
cd python
curl -O http://www.cotswoldjam.org/downloads/2015-04/candle.tgz
tar xvzf candle.tgz
```

…where -O is a minus sign followed by a capital letter O (not the numeral zero).