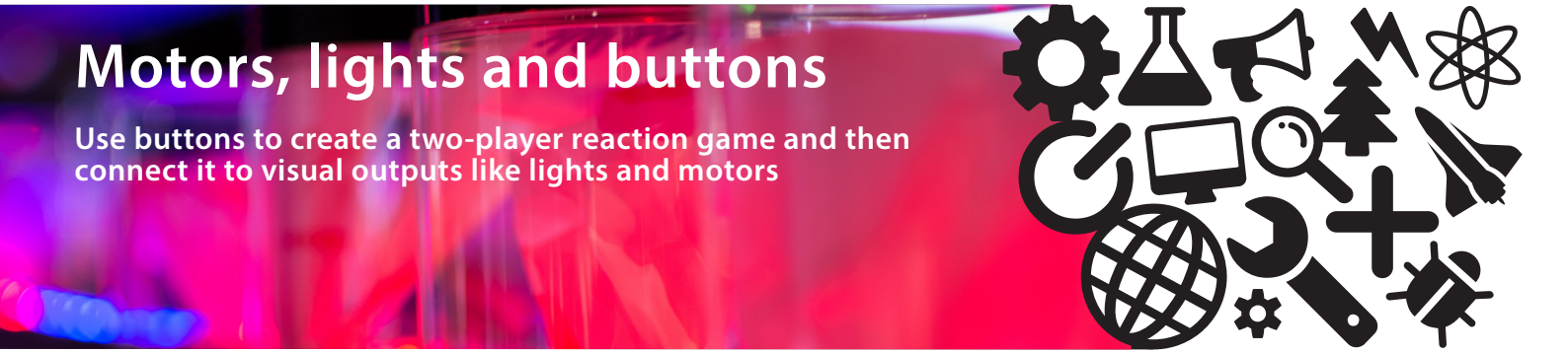
nustem

Motors, lights and buttons

Use buttons to create a two-player reaction game and then connect it to visual outputs like lights and motors



Ingredients

Platform:	Raspberry Pi / Raspbian Python3
Components:	ExplorerHat Motor Buttons
Libraries:	gpiozero, time explorerhat, random

Motors

If you want to make something move with your Pi, you're going to need a motor or a servo. In this example we're using motors as part of a two-player reaction game.

You can't connect a motor directly to the GPIO pins on the Pi because they can't provide enough current to run it. The ExplorerHAT contains motor driver circuits which provide enough oomph, without risking damage to the Pi. The ExplorerHAT also has some controllable flashy lights, and touch inputs, and a few other niceties.

motor.py

All the code for this worksheet is in `Digital-Making/examples/reaction-game/`. Start by loading `motor.py` in Thonny.

Run the code, and both the motors should spin at their full speed for 5 seconds and then stop.

See if you can work out how to spin one of the motors backward instead of forward. What about forward for a few seconds and then backwards?

You can also change the speed of the motor, try:

```
explorerhat.motor.one.backward(50)
```

countdownlights.py

Load this example into Thonny. We're going to create a visual countdown on the ExplorerHAT using the 4 coloured LEDs. Work through the code so you're reasonably comfortable with it, then run it.

The lines which turn the lights on and off are fairly easy to spot:

```
explorerhat.light.yellow.on()
explorerhat.light.blue.off()
```

The `sleep` function allows us to delay the program for a while. Because I want some suspense in my game, I've chosen a random delay between each light going out. I'm generating a random number between 1 and 5 seconds.

```
sleep(random.uniform(1,5))
```

reactiongame.py

Let's play a game. Load `reactiongame.py` in Thonny.

This lets us play a reaction game with two people. The person with the quickest reactions scores a point in each round. The winner is the first to 3.

Play the game.

The code uses the ExplorerHAT library to set up and control the buttons and then prints information about the game into the terminal. Go through the code and see if you can work out how the game will play: what will you seen in the terminal window, how will you know when to press the button, who has the highest score, and who won?

Now run the code and play the game with your partner!

When the fun stops, stop.

Take a closer look at the function totaliser:

```
def totaliser(player):
    print('P%d was first' % player)
```

The `%d` allows us to include a variable in a string in the `print` function. In this case the `%d` allows us to include an integer; we could have used `%s` to incorporate a string of text instead.

So if the script called:

```
totaliser(1)
```

The output in the terminal would be;

```
P1 was first
```

Now look back at the code and try to work out how we check if we have a winner.

Exercise: Pulling it all together

We're going to combine our flashy lights, motors and buttons to create the best game ever. Instead of printing everything to the terminal, we'd like to use the lights to display the countdown and the motors to show us when we have a winner. You can take the code from the previous python scripts.

First work out where you need to put the code and then copy and paste it in place. When you think you've got everything you need, run the code. The lights should indicate when you need to press the button and the motors will float the paper person of the winner.

If you're looking for a possible solution, open up `theFullGame.py`.