Week 6: Inputs & Interaction

Part A: Introduces concepts and examples of events and their use within p5.js.

Part B: Introduces a technical exercise using GUI Elements.

Part A: The concept of events

Events

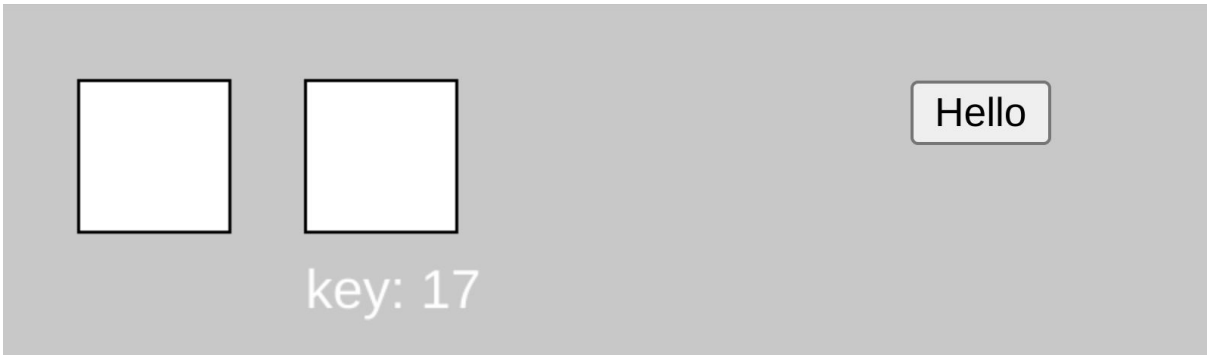
An event can include the movement of a mouse, a click on a key, or interaction with a mobile device. Events alter the normal flow of a program when an action such as a key press or mouse movement takes place. Key presses and mouse movements are stored until the end of draw() so won't disturb drawing that's currently in progress. The ‘events’ section of https://p5js.org/reference provides a list of events functionality. The following examples provide a basic introduction:

Setup

Create a new p5.js sketch with VScode and make sure it contains some variables that we will be using later in the sketch:

```
let r, g, b, x, y;

function setup() {
  createCanvas(400, 300);
  r = 200;
  g = 200;
  b = 200;
}

function draw() {
  background(r, g, b);
}
```

We can then write/add these examples into the draw function which show how different kinds of events work. Our final sketch will eventually contain these elements, where each shape performs a different event.



Capture mouse clicks

This determines if the mouse has been clicked anywhere on the canvas.

```
// Mouse pressed event - mouseIsPressed is a variable reserved by p5.js
// This changes a rect into an ellipse on a mouseclick
  push();
  fill(255);
  if (mouseIsPressed === true) {
    ellipse(50, 50, 50, 50);
  } else {
    rect(25, 25, 50, 50);
  }
  pop();
```

Trigger a keypress event

This determines if a key has been pressed - specifically the ‘r’ key. A variable specific to p5.js, called keyCode, contains a reference to the current key that has been clicked.

```
// Key pressed event.
// The keyCode variable identifies which key i.e d=68, f=70
  push();
  if (keyIsPressed === true && keyCode===82) { // if key is "r"
    fill(200,0,0);
  } else {
    fill(255);
  }
  textSize(18);
  text("key: "+keyCode, 100, 100);
  rect(100, 25, 50, 50);
  pop();
```

We can also create HTML elements in p5.js

For reference. See the ‘DOM’ section of https://p5js.org/reference. First create a button in the setup and attach a ‘myButtonClick’ function to it.

```
function setup() {
  createCanvas(400, 400);
  r = 200;
  g = 200;
  b = 200;
  myButton = createButton('Hello');     // Create a button
  myButton.position(300, 300);          // set its position
  myButton.mousePressed(myButtonClick); // On a MouseClick call myButtonClick();
  myButton.style('background', 'rgb(100,200,50)'); // HTML elements use CSS
}
```

Then add this new function to our sketch. This function is triggered when the is clicked.

```
function myButtonClick() {
  r = random(50, 255);
  g = random(50, 255);
  b = random(50, 255);
}
```

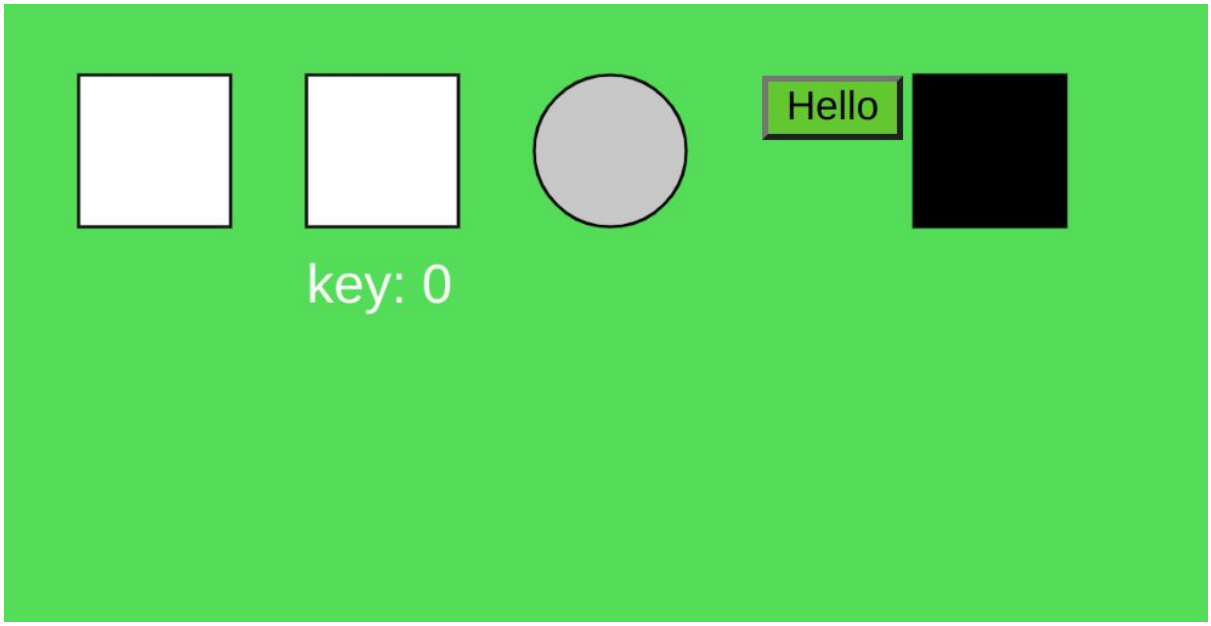
Note: HTML buttons created like this perform differently to p5 rectangles and ellipses because mouse and other events can be directly attached to them as per the example above. The draw back is that have to be styled using CSS (see https://www.w3schools.com/css/default.asp) rather than p5’s inbuilt functions.

The final code

```
function setup() {
  createCanvas(400, 300);
  r = 200;
  g = 200;
  b = 200;
  myButton = createButton('Hello');     // Create a button
  myButton.position(250, 25);           // set its position
  myButton.mousePressed(myButtonClick); // On a MouseClick call myButtonClick();
  myButton.style('background', 'rgb(100,200,50)'); // HTML elements use CSS
}

function draw() {
  background(r, g, b);

  // Mouse clicks
  push();
  fill(255);
  if (mouseIsPressed === true) {
    ellipse(50, 50, 50, 50);
  } else {
    rect(25, 25, 50, 50);
  }
  pop();

  // Key pressed event.
  push();
  if (keyIsPressed === true && keyCode===82) { // if key is "r"
    fill(200,0,0);
  } else {
    fill(255);
  }
  textSize(18);
  text("key: "+keyCode, 100, 100);
  rect(100, 25, 50, 50);
  pop();

}

function myButtonClick() {
  r = random(50, 255);
  g = random(50, 255);
  b = random(50, 255);
}
```

Advanced examples to experiment with

If you feel confident with the above then try playing with the following more advanced examples. They will produce a sketch that looks something like this:



Attach events to a circular area of the canvas

If the mouse is in the circle then

```
// Event on a circle
  push();
  x = 200;
  y = 50;
  let radius = 50;
  let distance = dist(mouseX, mouseY, x, y); //distance between mouse and circle
  if(distance < radius/2){ // If the mouse is in the radius
    fill(100);
    cursor(HAND);
  } else {
    fill(200);
    cursor(ARROW);
  }
  ellipse(x, y, radius, radius);
  pop();
```

Attach events to rectangle areas of the canvas

```
// Event on a rectangle
  push();
  x = 300;
  y = 25;
  w = 50;
  h = 100;
  fill(255);
  // Works out the coordiantes on the screen
  if ((mouseX>x) && (mouseX<x+w) && (mouseY>y) && (mouseY<y+h)){
    fill(0);
  }
  rect(x, y, 50, 50);
  pop();
```

Quiz: where might you add the following to the example above to make the rectangle clickable?

```
&& mouseIsPressed === true
```

Trigger events when the window is resized

Adding this function to the sketch will cause the background to change on resizing the window.

```
function windowResized() {
  r = random(50, 255);
  g = random(50, 255);
  b = random(50, 255);
  resizeCanvas(windowWidth, windowHeight);
}
```

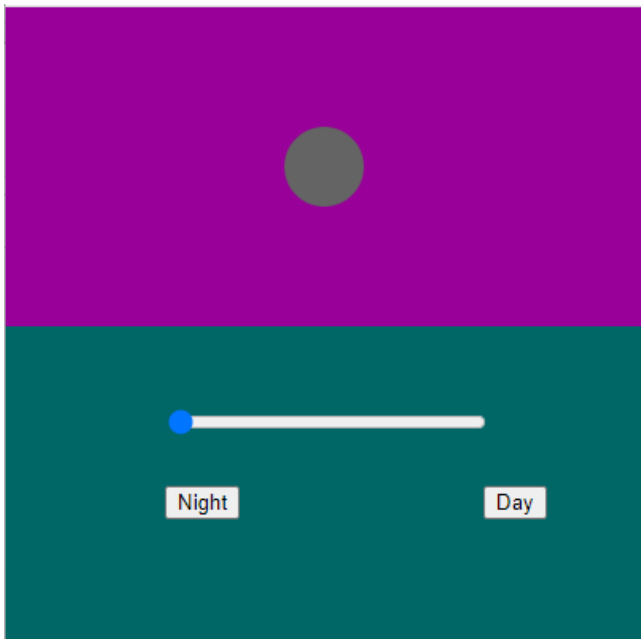
The complete code with extra examples

```
let r, g, b, x, y;

function setup() {
  createCanvas(400, 300);
  r = 200;
  g = 200;
  b = 200;
  myButton = createButton('Hello');     // Create a button
  myButton.position(250, 25);           // set its position
  myButton.mousePressed(myButtonClick); // On a MouseClick call myButtonClick();
  myButton.style("");
}

function draw() {
  background(r, g, b);

  // Mouse clicks
  push();
  fill(255);
  if (mouseIsPressed === true) {
    ellipse(50, 50, 50, 50);
  } else {
    rect(25, 25, 50, 50);
  }
  pop();

  // Key pressed event.
  push();
  if (keyIsPressed === true && keyCode===82) { // if key is "r"
    fill(200,0,0);
  } else {
    fill(255);
  }
  textSize(18);
  text("key: "+keyCode, 100, 100);
  rect(100, 25, 50, 50);
  pop();

  // Event on a circle
  push();
  x = 200;
  y = 50;
  let radius = 50;
  let distance = dist(mouseX, mouseY, x, y); //distance between mouse and circle
  if(distance < radius/2){ // If the mouse is in the radius
    fill(100);
    cursor(HAND);
  } else {
    fill(200);
    cursor(ARROW);
  }
  ellipse(x, y, radius, radius);
  pop();
```

```
  // Event on a rectangle
  push();
  x = 300;
  y = 25;
  w = 50;
  h = 100;
  fill(255);
  // Works out the coordiantes on the screen
  if ((mouseX>x) && (mouseX<x+w) && (mouseY>y) && (mouseY<y+h)){
    fill(0);
  }
  rect(x, y, 50, 50);
  pop();

}

function myButtonClick() {
  r = random(50, 255);
  g = random(50, 255);
  b = random(50, 255);
}

function windowResized() {
  r = random(50, 255);
  g = random(50, 255);
  b = random(50, 255);
  resizeCanvas(windowWidth, windowHeight);
}
```

Part B: Technical Exercise with GUI Elements

For this week's exercise, we will create a simple sketch that uses GUI elements to control the rising and setting of the sun and moon.



Step 1: Variables

At the top of your sketch, declare 1 variable to store the angle of the sun and moon.

```
let angle = 0; // variable to store the angle of the sun/moon
```

Step 2: Define the Setup function

Define a setup function to create a canvas with a width of 400 and height of 400. As we will be manipulating the angle of the sun and the moon, set the rotation mode to degrees.

I have also set `noStroke()` so there are no outlines in the sketch.

```
// the setup function runs once at the beginning
function setup() {
  // create a canvas on the screen of a set size
  createCanvas(400, 400);

  // set the rotation values to be in degrees
  angleMode(DEGREES);

  // no outline on shapes
  noStroke();



}
```

Step 3: Define the Draw function

Next, define a draw function and set the background to a colour for the sky.

Draw a rectangle at the bottom of the canvas as the foreground.

```
/* the draw function loops until the program is closed */
function draw() {

  // draw the background for the sky colour
  background(153, 0, 153);

  // draw a rectangle for the foreground
  fill(0, 102, 102);
  rect(0, 200, 400, 250);


}
```

Step 4: Draw the Sun

To draw a rotating sun we are going to use a `push() | pop()` section. This allows us to rotate the sun around the centre of the canvas, so it appears to rise and set. As we are rotating around the centre of the canvas, inside the push | pop section we set the translate() function to the draw from the centre:

`translate(width/2, height/2);`

Again in the push | pop section, the rotate() function is used to change the rotation. It's important to remember that the rotation is around the centre of the canvas. The variable `angle` is used to set the rotation.

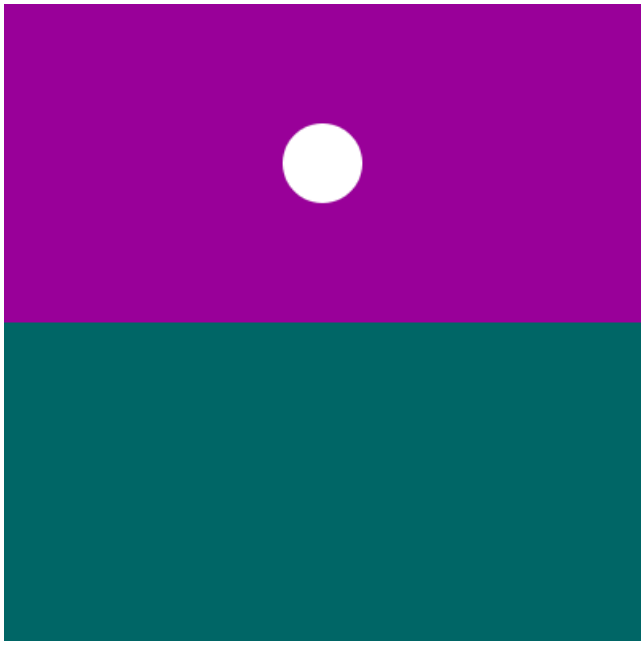
`rotate(angle);`

Then, still inside the push | pop section draw the sun using the ellipse function with a fill colour of your choice.

```
fill(255);
ellipse(0, -100, 50, 50);
```

To test if the movement is correct we can write a line to update the angle variable in the draw function (after the push | pop section) on every loop: `angle += 0.5;` Here the 0.5 value is the speed of rotation. This is just to test the movement, this will be deleted later as we will be controlling this with buttons and sliders.

```
push();

      translate(width/2, height/2); // set the drawing position to the centre
      rotate(angle); // set the angle of the drawing position

      // draw the Sun
      fill(255);
      ellipse(0, -100, 50, 50);

pop();

angle += 0.5; // test the suns movement
```

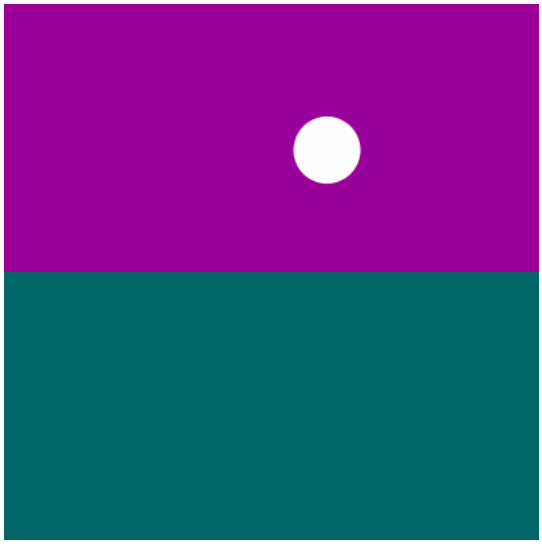


Step 5: Draw the Moon

We can add the moon into the same push | pop section. As long as the rectangle for the foreground is drawn after the sun and moon will appear to rise and set behind it. Add the following lines **inside the push | pop section**:

```
// draw the Moon
fill(100);
ellipse(0, 100, 50, 50);
```

For the moon, choose a fill of your choice.



Step 6: Create a ‘Day’ Button

We are going to use GUI buttons to control whether it is day or night. Buttons are created in the **setup function**. Starting with the day, we create a new variable to store the button - here it is called `buttonDay`

Then we use the `createButton()` function to generate a button. In the brackets is what will be printed on the button.

```
buttonDay = createButton('Day');
```

To set the position of the button we write the following, with the x and y coordinates of the button in the brackets:

```
buttonDay.position(300, 300);
```

Finally, we specify what will happens when the button is pressed with the mouse. Here, we are going to use our own function called ‘theDay’ which we are going to define ourselves below. Notice that the function name is in the brackets.

```
buttonDay.mousePressed(theDay);
```

Now our setup function looks like this:

```
// the setup function runs once at the beginning
function setup() {
  // create a canvas on the screen of a set size
  createCanvas(400, 400);

  // set the rotation values to be in degrees
  angleMode(DEGREES);

  // no outline on shapes
  noStroke();

  buttonDay = createButton('Day');  // create a button for the day, display the text 'Day' on it
  buttonDay.position(300, 300); // set the position of the button
  buttonDay.mousePressed(theDay); // set the function to be called when the button is clicked

}
```

Next is to create our own function ‘theDay’ so whenever the button is clicked the sun rises!

Step 7: Create a function ‘theDay’

We can create our own function in a similar way to how we define the setup and draw functions. Custom functions are defined below the setup and draw functions.

```
function theDay() {
  //code that runs when the function is called anywhere
}
```

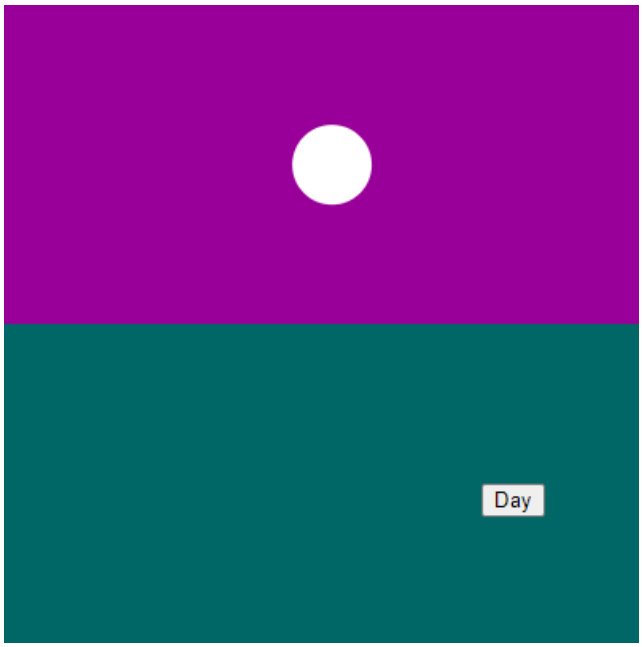
Inside the function we want to set the angle variable to the degree that the sun has risen and the moon set. So inside theDay function we set the angle to:

```
angle = 360;
```

This means every time the ‘Day’ button is clicked the angle will change to 360 and rotate the sun and moon accordingly.

```
/* function called when the day button
is pressed. Sets the angle to position
the sun at the top */
function theDay() {
  angle = 360;
}
```

At this point we can delete the line that reads: `angle += 0.5;` in the draw function, as we are now controlling the angle with buttons.



Step 8: Repeat for a ‘Night’ Button and function

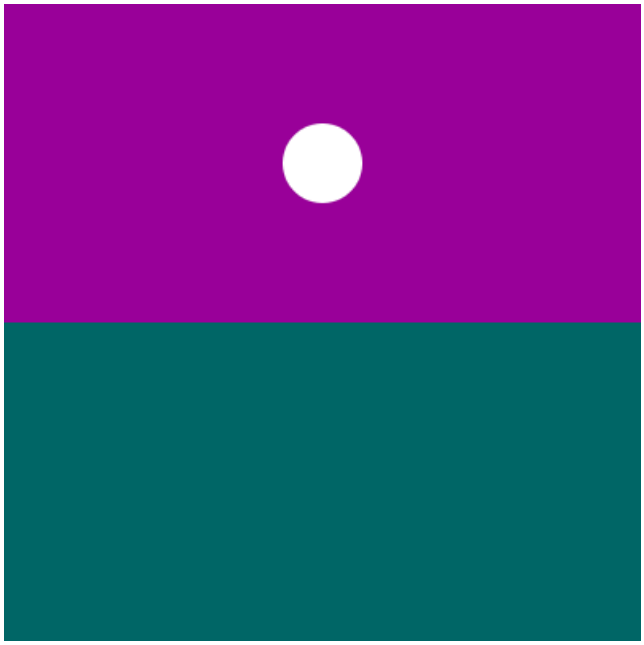
Create a ‘Night’ button in the same way. Add the following lines to the **setup function**. Notice the differences:

- The variable storing the button is called buttonNight
- The button is now set to read ‘Night’
- The position of the button is (100, 300)
- The function that is called when the button is called is ‘the Night’ (which we will define next)

```
// write this code inn the setup function
buttonNight = createButton('Night'); // create a button for the night, display the text 'Night' on it
buttonNight.position(100, 300); // set the position of the button
buttonNight.mousePressed(theNight); // set the function to be called when the button is clicked
```

Now we have to create a function called theNight to set the angle so the moon has risen and the sun has set. We define this below the theDay function at the end of the sketch. This function is called theNight and contains the line: `angle = 180;`

```
/* function called when the night button
is pressed. Sets the angle to position
the moon at the top, and updates the slider */
function theNight() {
  angle = 180;
}
```



Test your sketch by clicking on the buttons.

Step 9: Create a slider

The last part is to control the sun and moon using a slider. A slider is created in a similar way to a button in the **setup function**.

```
slider = createSlider(180, 360, 180);
```

Here we've created a variable to store the slider and used the `createSlider()` function. Inside the brackets the first 2 arguments set the range, and the third argument set the starting value of the slider.

Set the position of the slider in the same way as a button:

```
slider.position(100, 250);
```

The last bit is different to a button as we must set the width of the slider using the style function:

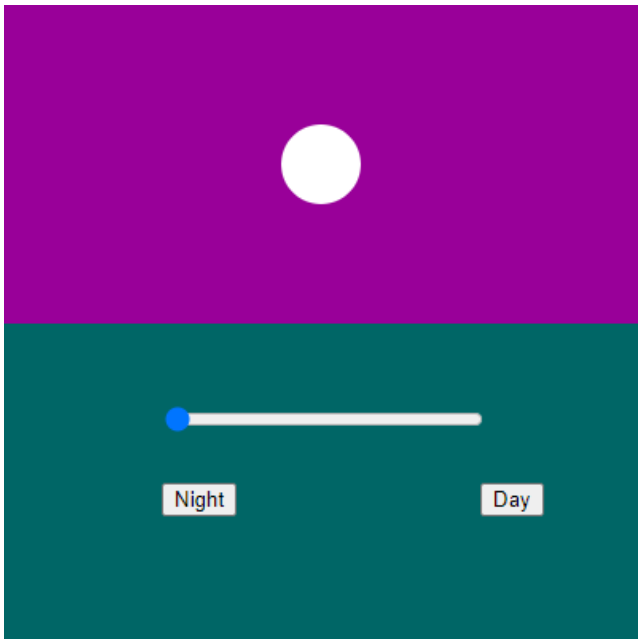
```
slider.style);
```

Note: This sort of styling work would usually be written in the CSS file but p5 allows us to do this in the sketch using the style function on GUI elements.

Now the full setup function should be as follows:

```
// the setup function runs once at the beginning
function setup() {
  // create a canvas on the screen of a set size
  createCanvas(400, 400);

  // set the rotation values to be in degrees
  angleMode(DEGREES);

  // no outline on shapes
  noStroke();

  buttonDay = createButton('Day');  // create a button for the day, display the text 'Day' on it
  buttonDay.position(300, 300); // set the position of the button
  buttonDay.mousePressed(theDay); // set the function to be called when the button is clicked

  buttonNight = createButton('Night'); // create a button for the night, display the text 'Night' on it
  buttonNight.position(100, 300); // set the position of the button
  buttonNight.mousePressed(theNight); // set the function to be called when the button is clicked
```

```
  slider = createSlider(180, 360, 180); // create a slider with range 180 to 360, set the slider to start at 180
  slider.position(100, 250); // set the position of the slider
  slider.style('width', '200px'); // set the width of the slider

}
```



Step 10: Set the angle with the slider value

Now we need to update the angle of the sun/moon with the slider value. We simply assign the variable angle to slider.value() at the bottom of the draw function:

```
angle = slider.value();
```

Drag the slider to set the angle of the sun and moon will rise and set accordingly.

The full draw function should read:

```
/* the draw function loops until the program is closed */
function draw() {

  // draw the background for sky colour
  background(153, 0, 153);

  /* draw 2 ellipses in a push and pop
   section to rotate around the middle
   of the canvas */
  push();

    translate(width/2, height/2); // set the drawing position to the centre
    rotate(angle); // set the angle of the drawing position

    // draw the Sun
    fill(255);
    ellipse(0, -100, 50, 50);

    // draw the Moon
    fill(100);
    ellipse(0, 100, 50, 50);

  pop();


  // draw a rectangle for the foreground
```

```
  fill(0, 102, 102);
  rect(0, 200, 400, 250);

  // update angle of sun/moon with slider
  angle = slider.value();

}
```

We are nearly there. The only bit remaining is to update the slider if we press either of the buttons. To do this we must add a line to each of our custom functions updating the slider to the correct values for day (360) and night (180) which corresponds to the angle the sun and moon are set to.

In the theDay() function we add the line:

`slider.value(180);`

This sets the value of the slider using code rather than moving the slider.

Likewise, we add the following line to theNight() function as the angle when the moon is risen is 360:

`slider.value(360);`

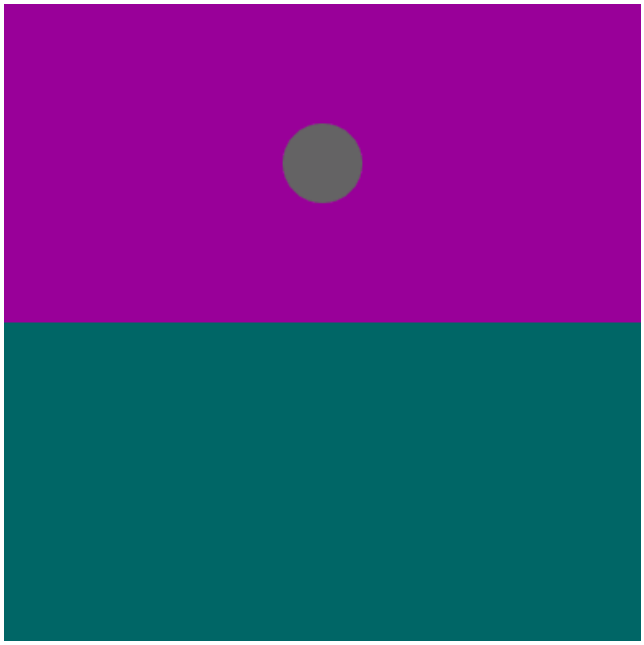
Now our custom function will read as follows:

```
/* function called when the day button
is pressed. Sets the angle to position
the sun at the top, and updates the slider */
function theDay() {
  angle = 360;
  slider.value(360);
}

/* function called when the night button
is pressed. Sets the angle to position
the moon at the top, and updates the slider */
function theNight() {
  angle = 180;
  slider.value(180);
}
```

Test it out to make sure it works!

Coding Challenge

Can you set the sky to change colour with the day and night? HINT: one way is to use the map() function we worked with last week to use the angle value to set the background colour.



Advanced Challenge: Can you code the buttons to rotate the sun and moon in a smoother way- so they rise and set smoothly like when manipulating the slider, instead of the sun and moon jumping into position instantly?