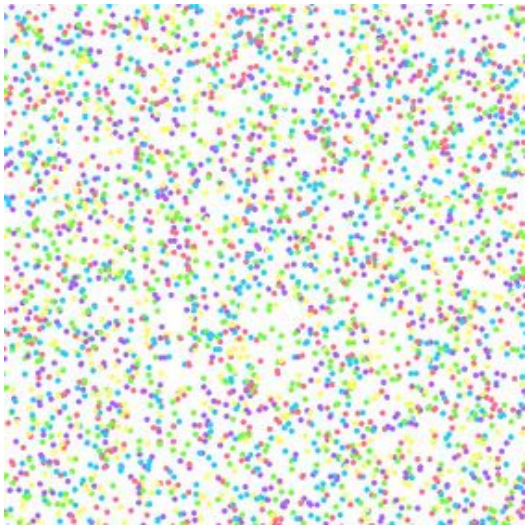
Keyboard and Mouse Inputs

So let's talk this week about conditional statements that will help give more structure to our sketches. Let's say on `keyPressed()` we want to draw 20 ellipses to the screen.



```
color[] colorList =
{color(168,100,253),color(41,205,255),color(12
0,255,68),color(255,113,141),color(253,255,106
)};

void setup(){
  size(900,900);
  background(255);
  noStroke();
}

void draw(){
}

void keyPressed(){
  for(int i =0; i < 20; i++){
    fill(colorList[int(random(5))]);
    ellipse(random(width),random(height),
10,10);
  }
}
```

This should be fairly familiar to you. Now let's say we want to define what happens when the 'm' key is pressed. The first thing we're going to need is a conditional statement known as an `if` statement.

If/Else statement

So some of you may have already ventured into the land of `if` statements but if you haven't don't worry - it's fairly straight forward. We want the computer to do the following:

1. On key press, run the function `keyPressed()` and…
2. ...`if` the key is 'm'(for magic), execute the following statements
3. ...`else`, execute the following statements instead.

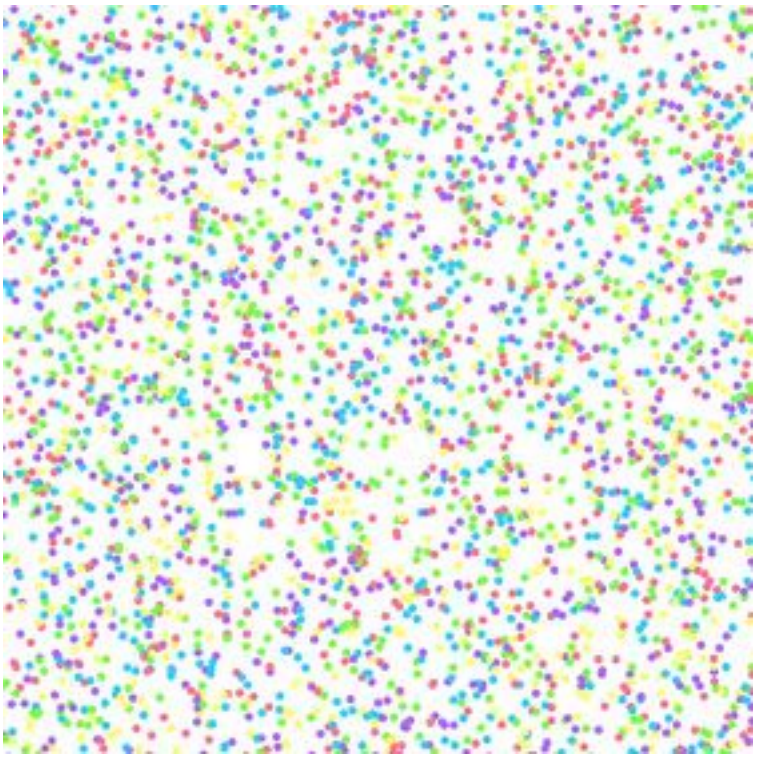
This is represented like this:

```
void keyPressed(){
  if (key == 'm'){
    //Execute these statements
  } else {
    //Execute these statements
  }
}
```

You can even chain it like this:

```
void keyPressed(){
  if (key == 'm'){
    //Execute these statements
  } else if (key == '1'){
    //Execute these statements
  } else {
    //Execute these statements
  }
}
```

Can you make this work in the previous code?
We only want the key 'm' to trigger the drawing of the 20 colourful confetti.



```
color[] colorList =
{color(168,100,253),color(41,205,255),color(120,255,68),color(255,113,141),color(253,2
55,106)};

void setup(){
  size(900,900);
  background(255);
  noStroke();
}

void draw(){
}

void keyPressed(){
  if (key == 'm'){
    for (int i =0; i < 20; i++){
      fill(colorList[int(random(5))]);
      ellipse(random(width),random(height), 10,10);
    }
   }
}
```

technecolour

Relational Operator

Another thing you may have noticed is that there are TWO equal signs.

```
...
  if (key == 'm'){
...
```

This is what's known as a Relational Operator; an operator that checks if items are *related* to one another. In this case, we're asking the computer to check the "equality" of `key` and the letter `'m'`. Some other Relational Operators you might recognise are:

a. != (inequality)
b. < (less than)
c. <= (less than or equal to)
d. > (greater than)
e. >= (greater than or equal to)

Change the equality operator to the inequality operator in your sketch.
Now change it back to equality.

Logical Operators

So given we're checking for == (equality), the key `m` is not the same as the key `M` to the computer. Let's first see what it's doing. **Run your sketch again and type 'm' and then 'M'.**

If we want to trigger our colours on `m` or `M`, or any other key in fact, then we will need a logical operator. Instead of just checking for `m`, we want to check for `m` OR `M`.

We could do the following:

```
void keyPressed(){
  if (key == 'm'){
    //Execute these statements
  } else if (key == 'M'){
    //Execute these statements
  } else {
    //Execute these statements
  }
}
```

However, if we want 'm' and 'M' to execute the same statements, it's a waste of breath for the computer. So, we bring in a logical operator...

|| (OR)

If the key is `m` OR `M`:

```
...
if (key == 'm' || key == 'M'){
  //Execute these statements
}
...
```

! (NOT)

If the key is NOT `m` or NOT `M`:

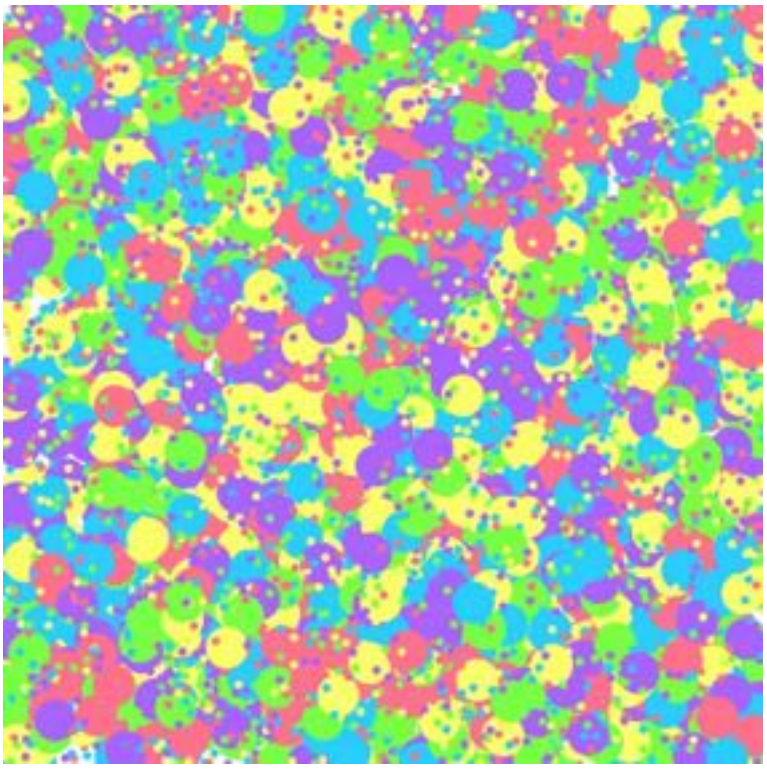
```
...
if (!(key == 'm') || !(key == 'M')){
  //Execute these statements
}
...
```

&& (AND)

If the key is `m` AND the canvas width = `900`

```
...
if (key == 'm' && width == 900){
  //Execute these statements
}
...
```

Try these different variations in your sketch. **Can you get 'm' OR 'M' to draw confetti and 's' OR 'S' to change the size of the confetti?**



```
//Can you get ‘m’ OR ‘M’ to draw confetti and ‘s’ OR ‘S’ to change the size of the confetti?
```

technecolour

Special Keys

The last missing piece of the puzzle is for situations where you want to use keys that aren't letters, numbers or punctuation. We have to be a little careful here as depending on where you run the code also depends on what else the system might execute with that "special" key.

Some keys require `key` and some require `keyCode` checks:

value	**variable check**
a-z, A-Z	key
" " (space)	key
ENTER	keyCode
BACKSPACE	keyCode
ESC	keyCode
UP, DOWN, LEFT, RIGHT	keyCode
ALT	keyCode
CONTROL	keyCode
SHIFT	keyCode

So for the 'special' keys that need `keyCode`. We'd do the following:

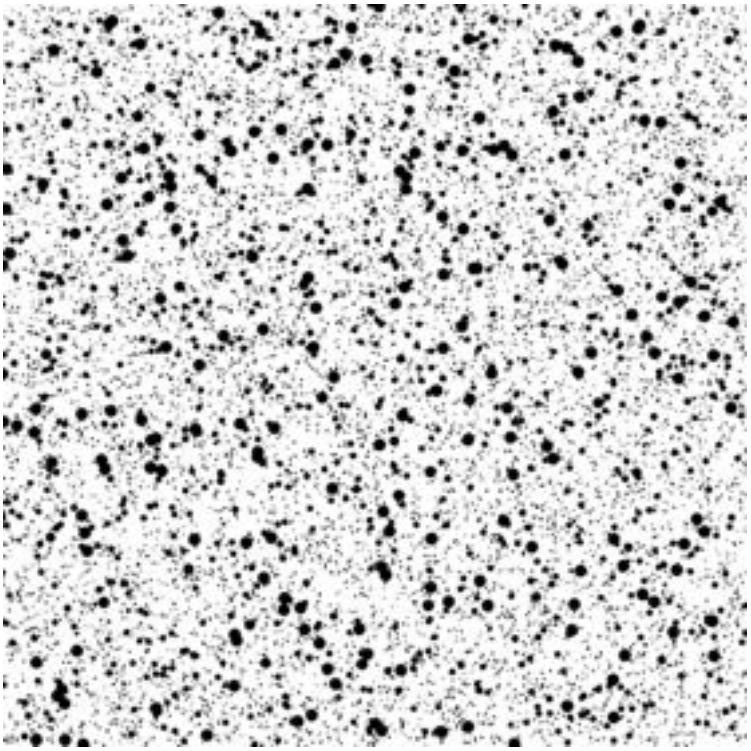
```
void keyPressed(){
  background(colorList[int(random(5))]);
  if (keyCode == UP){
    //execute these statements
  } else {
    //execute these statements
  }
}
```

Use the ESC key to draw confetti.
What's the problem in Processing? Now try this in openProcessing, what do you notice now?

Use the Space key to draw confetti.
Did you notice anything different here?

Let’s look at some practical uses.

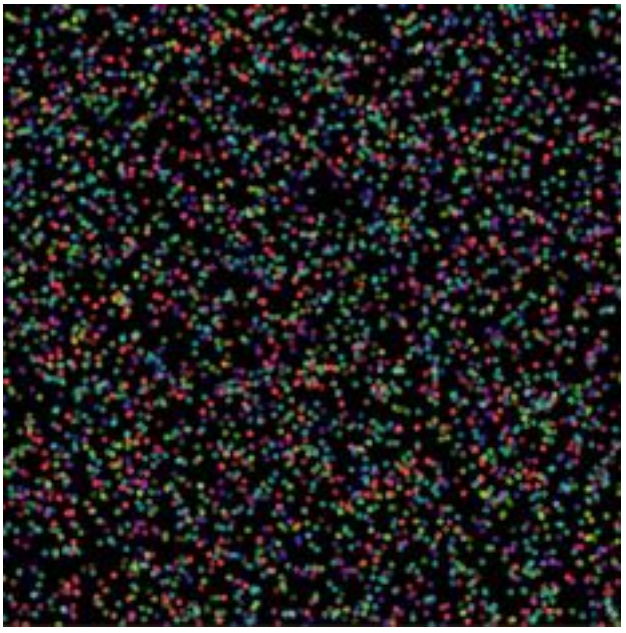
Multiple keys for different uses.



```
int ellipseSize = 2;

void setup(){
  size(900,900);
  background(255);
  noStroke();
}

void draw(){
  fill(0);
  ellipse(random(width),random(height), ellipseSize,ellipseSize);
}

void keyPressed(){
  if (key == ' '){
     for (int i = 0; i < 100; i++){
       ellipse(random(width),random(height), ellipseSize,ellipseSize);
     }
   } else if (keyCode == UP){
     ellipseSize++;
   } else if (key == 'r' || key == 'R'){
     ellipseSize = 2;
   }
}
```

Can you describe what’s happening here?
Add a new command: If a user presses to DOWN key, decrease the size of the ellipse.

Using arrow keys to increment and decrement color values

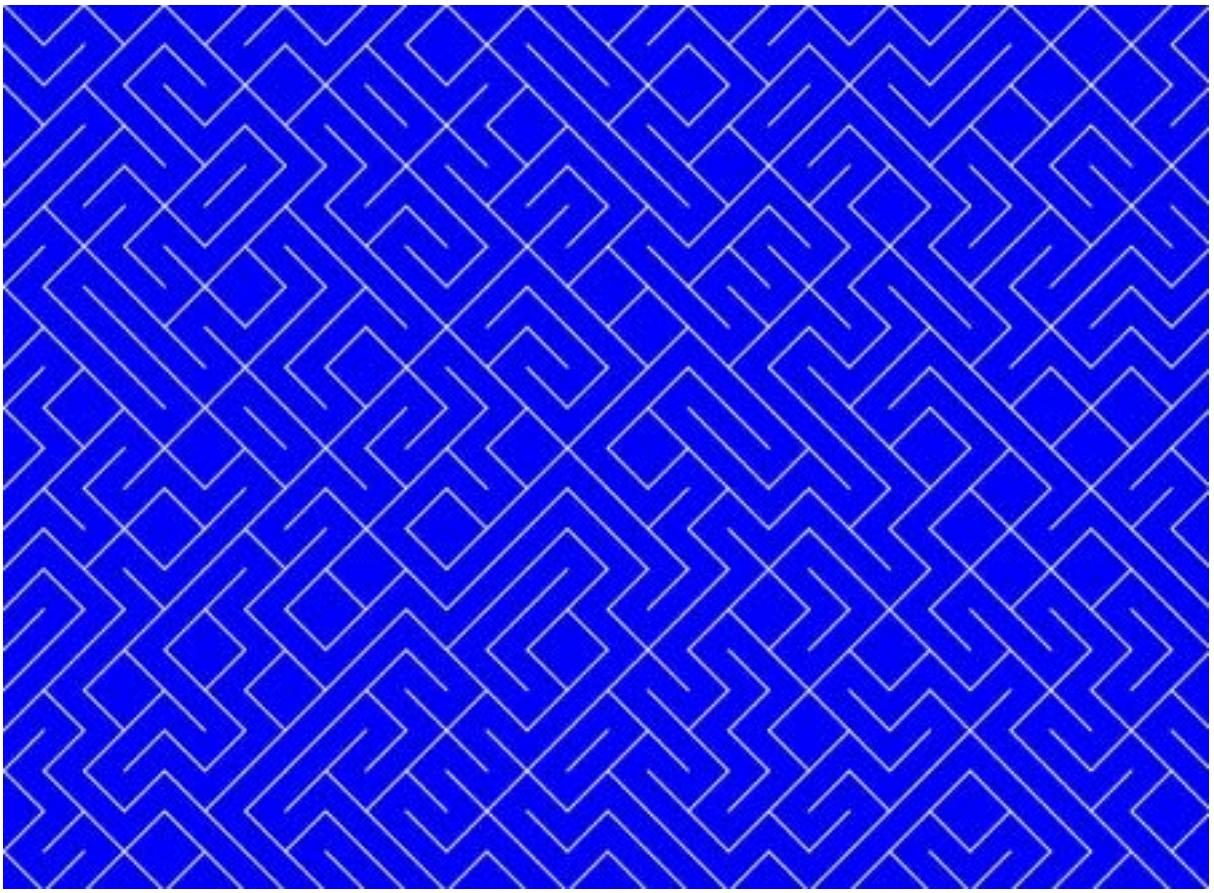


```
color currentColor;
int h = 359;
int t = 100;
boolean countUp = false;

void setup(){
  size(900,900);
  background(0);
  colorMode(HSB,360,100,100,100);
  noStroke();
}

void draw(){
  currentColor = color(h,75,90,t);
  fill(currentColor);
  ellipse(random(width),random(height), 10,10);
}

void keyPressed(){
  if (keyCode == UP && h >= -1 && h <= 359){
    h++;
  } else if (keyCode == DOWN && h >= 0 && h <= 360){
    h--;
  }

  if (keyCode == RIGHT && t >= -1 && t <= 99){
    t++;
  } else if (keyCode == LEFT && t >= 0 && t <= 100){
    t--;
  }
}

void mousePressed(){
  clear();
}
```

technecolour

Type a pattern

Can you work out what's going on here?



This concept is inspired by a single line of code written in BASIC(a programming language) on the Commodore 64(an early home computer) in the early 1980s:

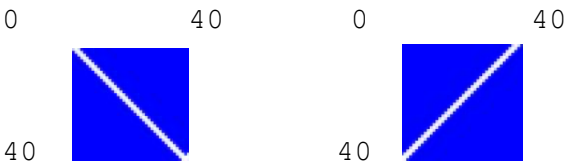
```
10 PRINT CHR$(205.5+RND(1)); : GOTO 10
```

There's a book on it and you can see it in action here.

There's also a coding video for this in Javascript - however Daniel Shiffman is a man of glorious natures and although the code is a little different, the fundamental structure is well explained.

Let’s break this down into smaller bits.

The unit of this grid is made of a single line with alternating directions. The first option from top left to bottom right and the second option from bottom left to top right.



Let’s first draw these two lines onto the canvas.

```
void setup(){
  size(1200,880);
  strokeWeight(2);
  background(0,0,255);
}

void draw(){
}

void keyPressed(){
  background(0,0,255);
  stroke(255);
  line(0,0,40,40);
  line(40,0,0,40);
}
```

Now rather than drawing them on top of one another, I want to draw one OR the other. So we need an `if` statement with a bit of probability.

```
void setup(){
  size(1200,880);
  strokeWeight(2);
  background(0,0,255);
}

void draw(){
}

void keyPressed(){
  background(0,0,255);
  stroke(255);
  if (random(1) < 0.5){
    line(0,0,40,40);
  } else {
    line(40,0,0,40);
  }
}
```

Now, what’s neat about this is that if we change the probability and skew it in the one direction, we can make all kinds of other neat patterns. **Change one line’s probability to 0.3.**

Time to repeat! Repeat!

```
int rowPos = 0;

void setup(){
  size(1200,900);
  background(0,0,255);
  stroke(255);
  strokeWeight(2);
}

void draw(){
}

void keyPressed(){
  if (random(1)<0.5){
    line(rowPos,0,rowPos+40,40);
  } else {
    line(rowPos+40,0,rowPos,40);
  }
  rowPos += 40;
}
```

Now we need a check at the end of the line, and create a new line.

```
int rowPos = 0;
int colPos = 0;

void setup(){
  size(1200,900);
  background(0,0,255);
  stroke(255);
  strokeWeight(2);
}

void draw(){
}

void keyPressed(){
  if (random(1)<0.5){
    line(rowPos,colPos,rowPos+40,colPos+40);
  } else {
    line(rowPos+40,colPos,rowPos,colPos+40);
  }
  rowPos += 40;

  if (rowPos > width){
    rowPos = 0;
    colPos += 40;
  }
}
```

What if we just wanted to have the pattern type out on it's own?
See if you can get this to work.

Mouse and key functions

Here are some more functions that might come in handy.

`keyPressed();`	Called once every time a key is pressed.
`keyReleased();`	Called once every time a key is released.
`mousePressed();`	Called once after every time a mouse button is pressed.
`mouseMoved();`	Called once every time the mouse moves while a mouse button is **not** pressed.
`mouseDragged();`	Called once every time the mouse moves while a mouse button **is** pressed.
`mouseClicked();`	Called after a mouse button has been pressed and then released

Week 6 Exercise

Make your **own** PRINT10 pattern using different line directions or shapes. See what surprising patterns you can create using different probabilities.

Utilise at least two of the above functions to control how your pattern is created.