UNIVERSITY
OF HULL

Department of Computer Science and Technology

Applicant Experience - Asteroids in Processing Sheet 1

Welcome to the University of Hull's Applicant Experience from the Department of Computer Science and Technology!
If you are reading this, it is because you are considering an exciting future in the field of Computer Science.
Throughout this experience we will be utilising the latest Raspberry Pi 4, developing a classic arcade game: Asteroids. By the end of this series of tutorials/tasks you will have made your very own arcade game in a language known as Processing.

This worksheet will cover some advanced concepts which you will no doubt encounter throughout your studies. The main goal of this is to introduce you to what is possible within Computer Science and begin to break down some of the mystery behind everyday technology.

This road may be long and bumpy, but hopefully you will find it exciting, interesting, and at points a little bit challenging. We can't just jump straight into making an Asteroids game, we have to build up to it, splitting the concepts into small bite-sized pieces that we can write, and try to get working, before tackling the bigger problems. As professional programmers, we do this all the time. Just like trying to get any monstrous project complete, it starts with a step.

What is Processing, and why are we using it?

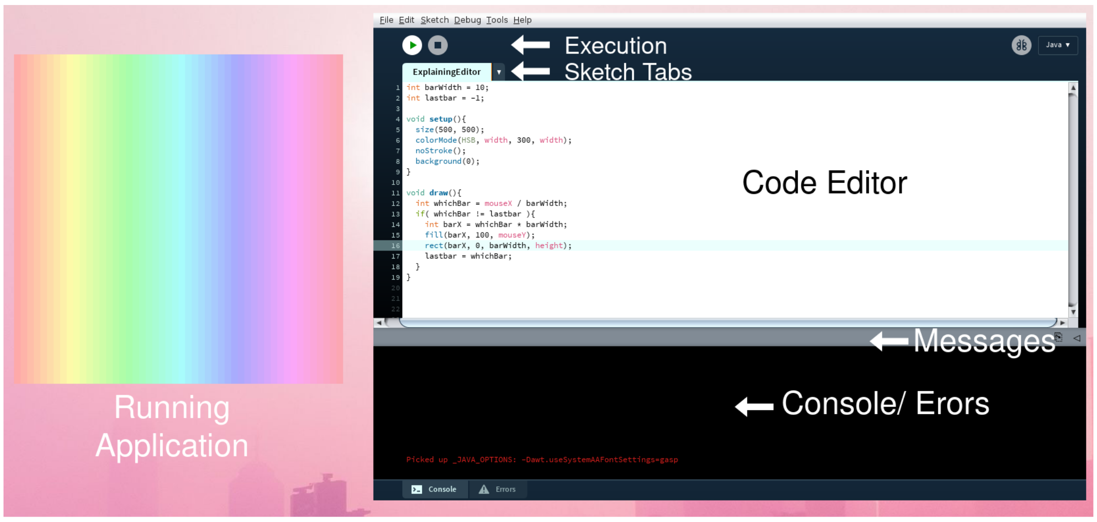
An excellent question! Processing is a computer programming language and overall ecosystem which is designed with feedback in mind. It enables us to very rapidly program concepts with visual components so we can 'see' the changes we're making. Although Processing is intrinsically bound to visuals, it provides a platform to implement many core concepts which are applicable to every field in Computer Science! Just because we're making a game, doesn't mean that it's limited to Games students; Computer Science and Software Engineering will also make use of these concepts.

Processing provides a Sketchbook interface where we can write some code, and execute our programs. It has extensive support for additional functionality via libraries (code other people have written for us to use!), and provides excellent documentation (for when we don't know what something is called). Both of these are incredibly useful for any program you write, something which you will become intimately familiar with when you begin your studies.

Grab your beverage of choice and a biscuit, and let's get started!

UNIVERSITY
OF HULL

The Processing Environment



```
int barWidth = 10;
int lastbar = -1;

void setup(){
  size(500, 500);
  colorMode(HSB, width, 300, width);
  noStroke();
  background(0);
}

void draw(){
  int whichBar = mouseX / barWidth;
  if( whichBar != lastbar ){
    int barX = whichBar * barWidth;
    fill(barX, 100, mouseY);
    rect(barX, 0, barWidth, height);
    lastbar = whichBar;
  }
}
```

```
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=gasp
```

To start the Processing IDE, you will find an icon on your Desktop called Processing. Alternatively, click in the top-left corner, go to graphics > Processing.

Running Application - This is the created window from our application. We define the size of this, and whatever code we write will use this to visualise to.

Code Editor - This editor is where we write our programming language
Messages - If we make any mistakes when writing our code, so that it won't run, the messages here will tell us what's wrong.

Console/Errors - Part of our program can output text to the console, this is useful if we wanted to know what value something is.

Execution - These two buttons are responsible for taking our code, and then running it (the 'play' button), and then similarly we can stop a running application by either closing the running application itself, or pressing the stop button.

Sketch Tabs - Writing everything in one big long file can get difficult. Oftentimes it is useful to split up what we're writing into multiple tabs (files).

For more information on the Processing Environment, there is excellent documentation on https://processing.org/reference/environment/

UNIVERSITY
OF HULL

First Steps

Our programs are called ‘sketches’ in Processing. When we save for the first time, we will be asked for a name. This will make a folder with all of our code neatly inside which can load back up later.

When we run our sketch there’s a few things we need to do first. We might want to specify a window size to create, set the background colour, etc. Processing has some code blocks which it will execute for us at the appropriate times.

The first example of this is called ‘Setup’, the code inside this is run once at the start of the program. This is useful as we can make our display window and create things which we’ll need to use later on.

Create a new Sketchbook by going to **File - New** (Control + N), your new sketch will be named similar to sketch_210521a, this is just the date.

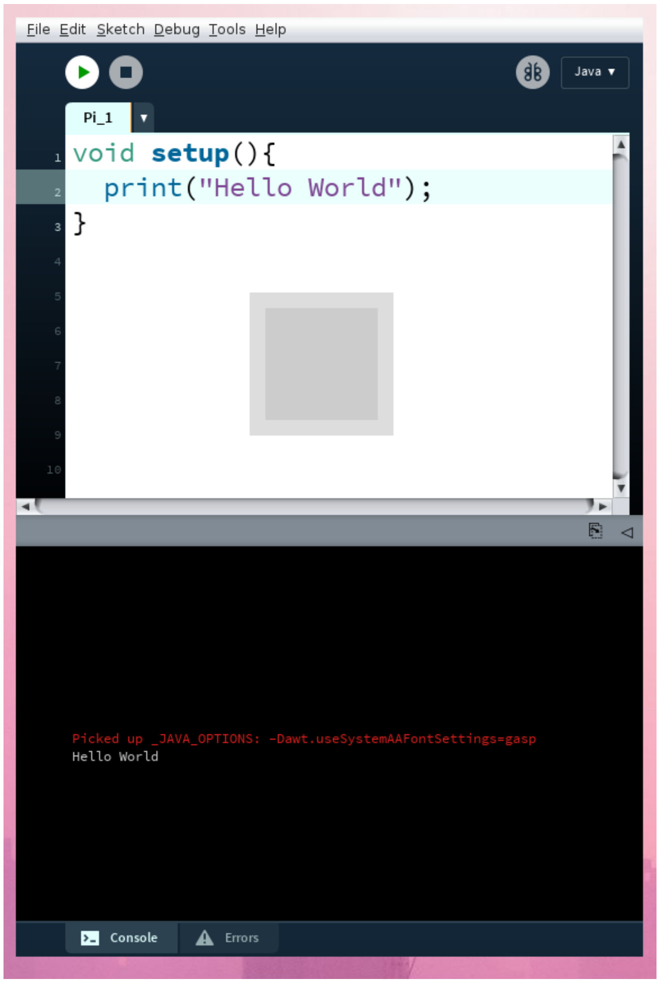
Write the following in this new blank sketch:

```
void setup(){
  print("Hello World");
}
```

Everything inside the { } brackets (curly brackets) is what will be executed when Processing first starts running our sketch. In our case, we just simply have one expression to run. `print()` accepts any text we pass into it, and will display it in our console. In our case, we would expect **Hello World** to show in the console.
Notice how for text we wrap it in “ “ marks (or sometimes ‘ ‘).

Run the sketch, and check the console area. You will also notice Processing has made a small default box window for us. This is because Processing expects us to do some fancy visuals, which we’ll get on to later.

You should see the following output:



```
void setup(){
  print("Hello World");
}
```

```
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=gasp
Hello World
```

You can try to put different text between the " " marks to get it to print whatever you want out to the console. Such as this classic internet meme.

Before we continue on, let's save our sketchbook. If we go to **File - Save** (Control + S) we should be prompted to choose a location and save. By default, Processing has its own folder for storing sketchbooks in. Let's name this **Pi_1** for now. The sketch_210521a name should now be replaced by **Pi_1**.

UNIVERSITY
OF HULL

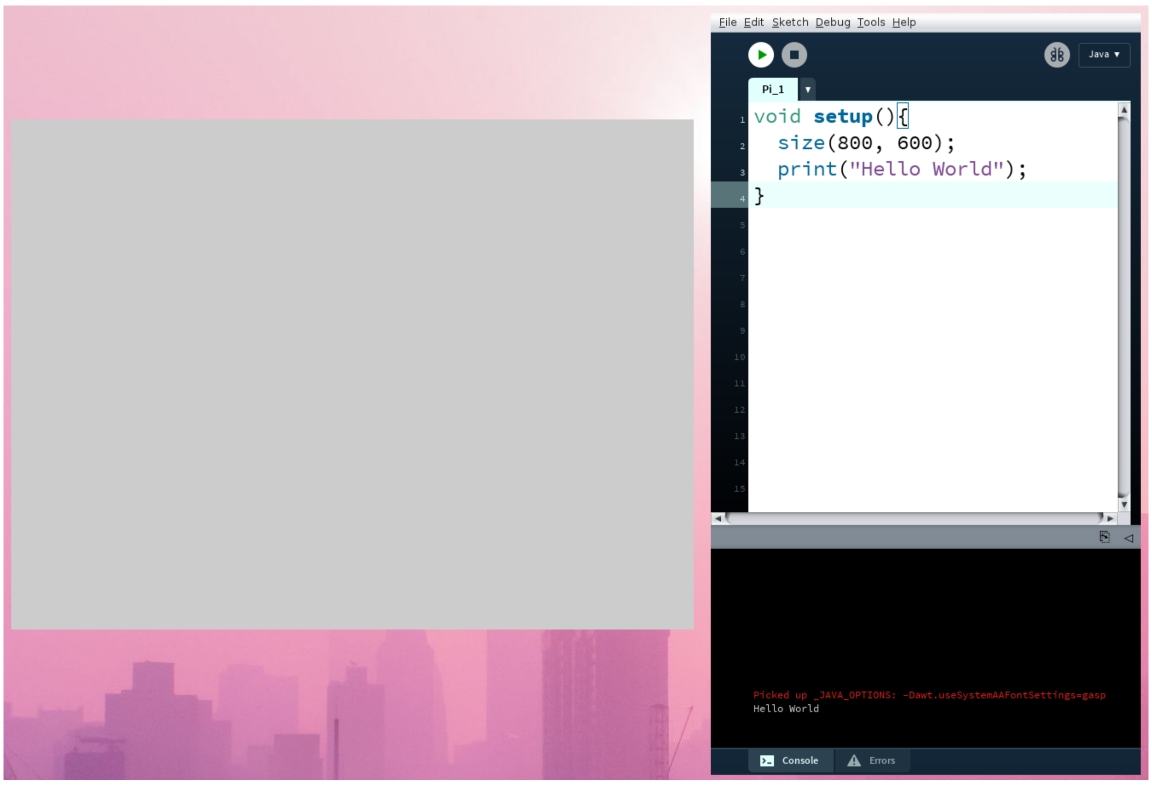
Working with Windows

Now that we can run a sketch, and see a small window, let's make it bigger. Inside the setup function, we can use size(width, height), to specify how big of a window we want for our sketch.

Let's add this to our code:

```
void setup(){
  size(800, 600);
  print("Hello World");
}
```

This will set up a window of size 800 pixels by 600 pixels, which is significantly larger than what we had before. Then it will print our Hello World text to the console area.



```
void setup(){
  size(800, 600);
  print("Hello World");
}
```

```
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=gasp
Hello World
```

If we wanted, we could put size(800, 600) after the print - but still within the { } brackets - and this would just change the order of what is executed. It would print, and then make a larger window.

NOTE: If you start experiencing slow-downs in your Processing Sketch running, as we ask it to do more and more, change your `size(800, 600);` to `size(800, 600, P2D);`. This specifically tells Processing to take advantage of a hardware accelerated renderer, which makes anything graphics related - like lots of things flying all over the place - much faster.

UNIVERSITY
OF HULL

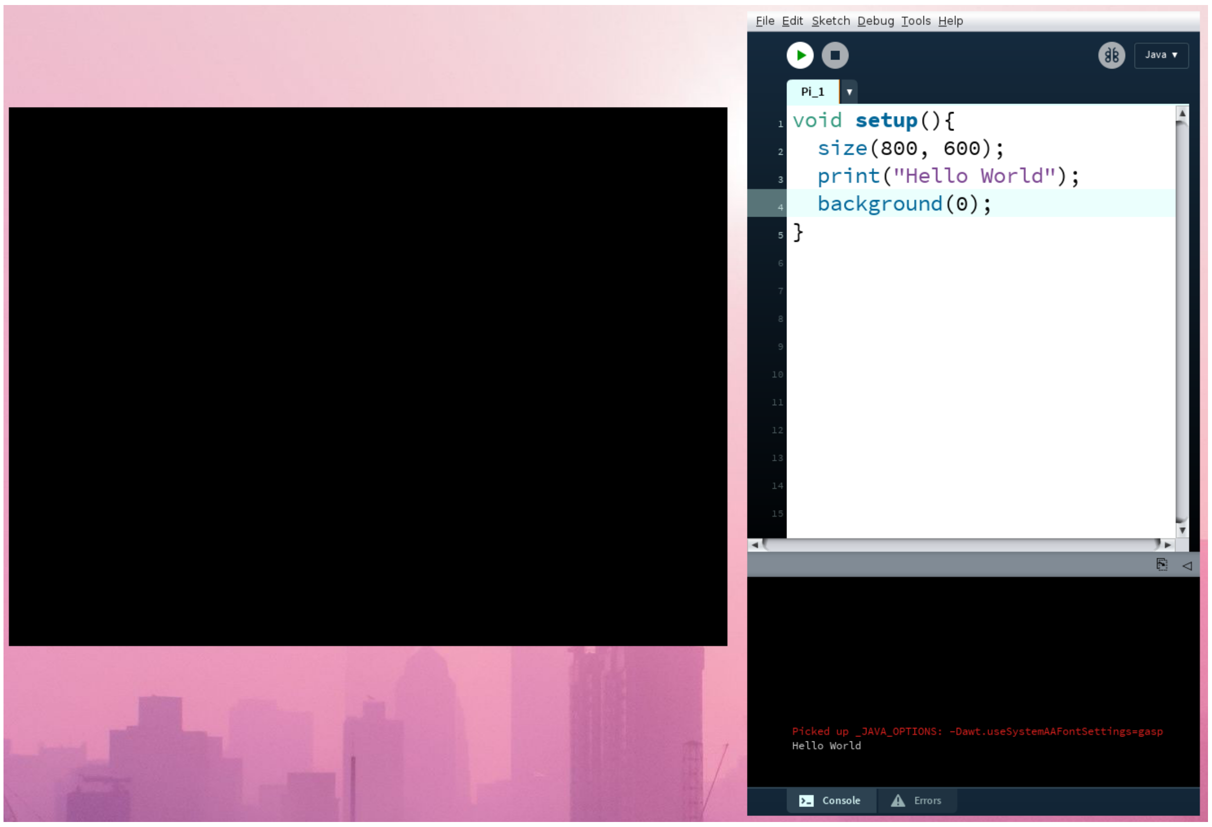
Add a splash of colour

Currently our window is bland and boring, with a grey colour throughout. Let's make things more vibrant. Processing allows us to change the background colour any time we want by use of a function called `background()` that accepts many values.

Anytime we are dealing with Processing functions, it's always a good idea to look at the Documentation provided by Processing. This tells us how to use these magical commands, and what to give them. In the case of background, this exists at https://processing.org/reference/background_.html

If we change our setup code to now look like the following, we should be able to make a black background.

```
void setup(){
  size(800, 600);
  print("Hello World");
  background(0);
}
```

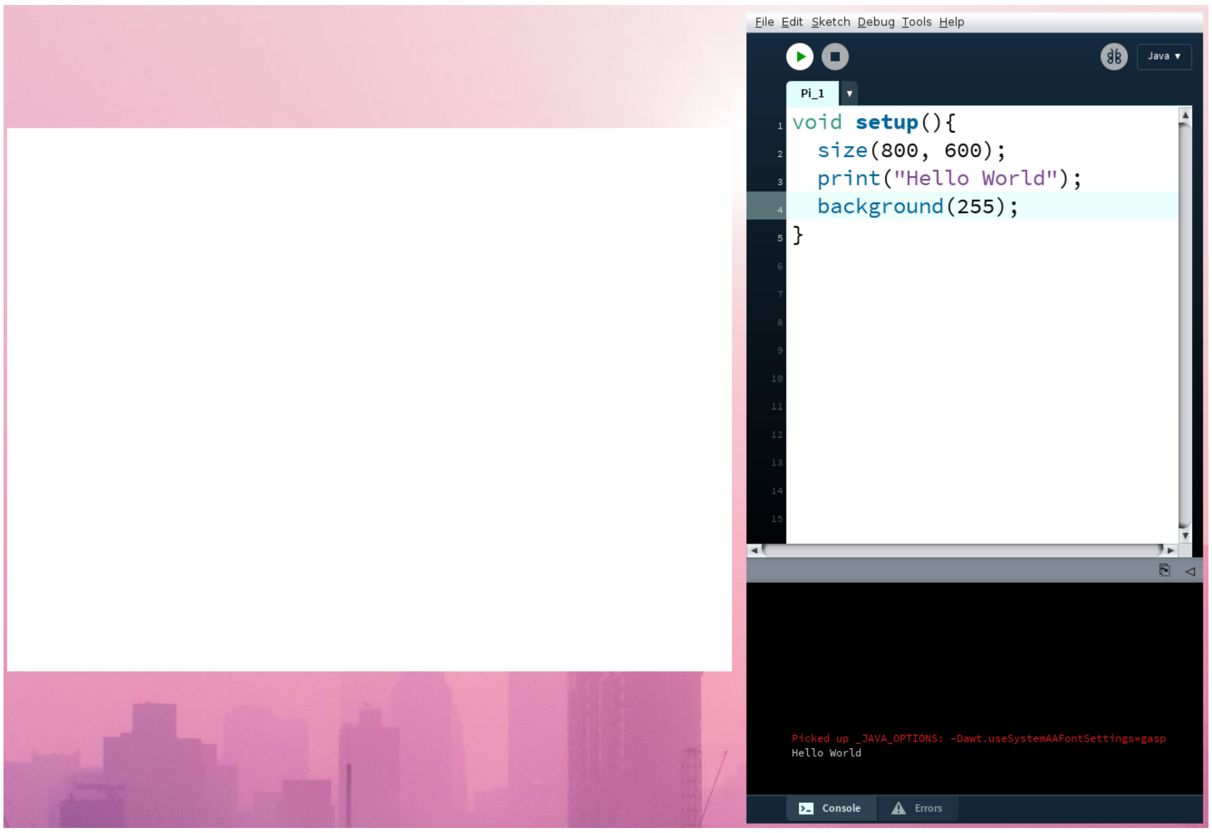


```
void setup(){
  size(800, 600);
  print("Hello World");
  background(0);
}
```

```
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=gasp
Hello World
```

UNIVERSITY
OF HULL

Try changing the background value from 0, to 255. In the case of a single number, Processing knows that 255 = pure white, and 0 = pure black. Values in between are all grey to some amount. E.g 127 would be middle grey.



```
void setup(){
  size(800, 600);
  print("Hello World");
  background(255);
}
```

```
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=gasp
Hello World
```

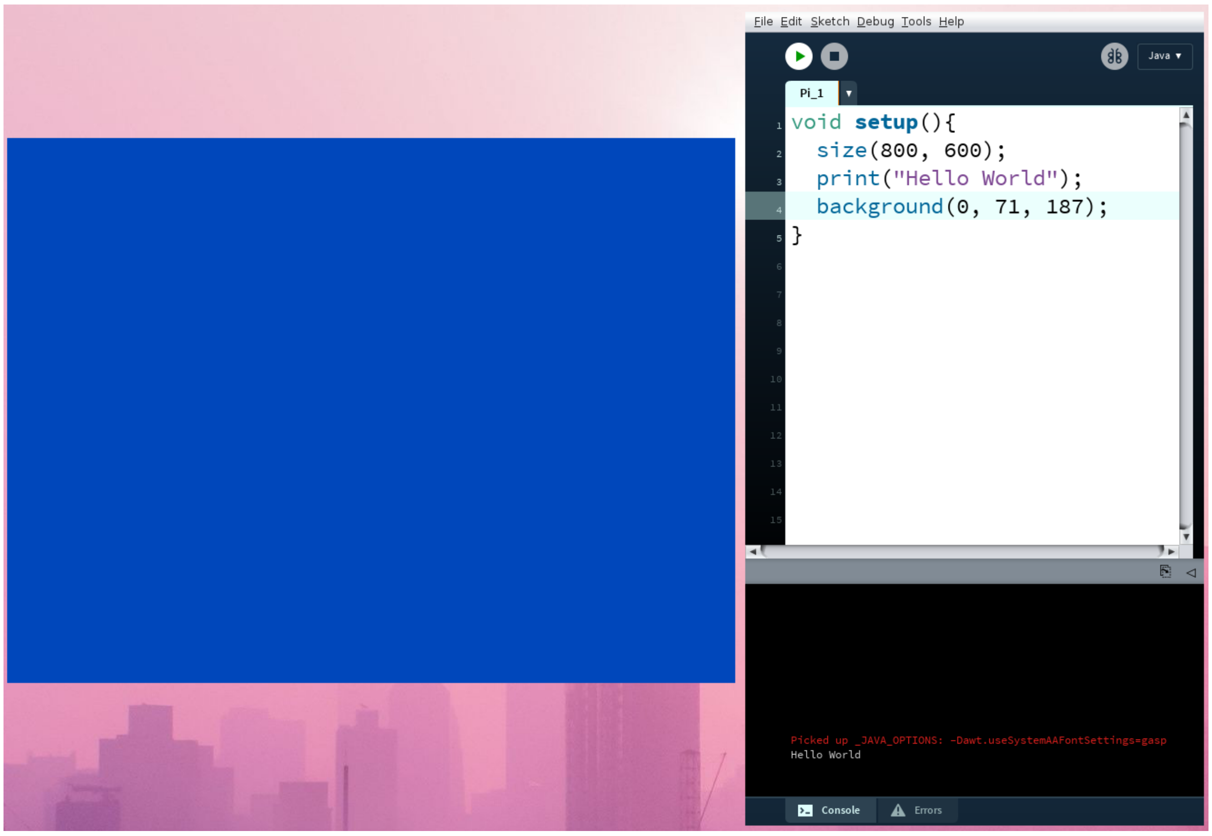
“But, this isn’t Colourful!” I hear you say. Let’s fix that. Background, instead of accepting just a single number, can accept multiple numbers. Typically when we deal with colours in computer science we use 3 numbers, and mix them together (like paint!). These are Red, Green, and Blue; collectively, often known as RGB.

For example one of our University colours is Pantone 2728 C (or Blue to you and me). If we look this up (https://www.pantone.com/uk/en/color-finder/2728-C) we can see that this specific colour has:

RGB 0 71 187

Each value here ranges from 0 to 255, just like our boring greys. This RGB value represents 0 Red, 71/255 Green, and 187/255 Blue. If we were to mix these three colours in these ratios together, we would get a type of blue. From the values you can see that the colour is made up of mostly blue, with a bit of green mixed in, and no red.

If we replace our `background(255);` with `background( 0, 71, 187 );` we should now have a lovely blue window to use.



```
void setup(){
  size(800, 600);
  print("Hello World");
  background(0, 71, 187);
}
```

```
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=gasp
Hello World
```

These same concepts for colours will crop up throughout, as we use the same RGB representation any time we want to change the colour of something. This can be our canvas background, shapes, text, any element that can be colourful.

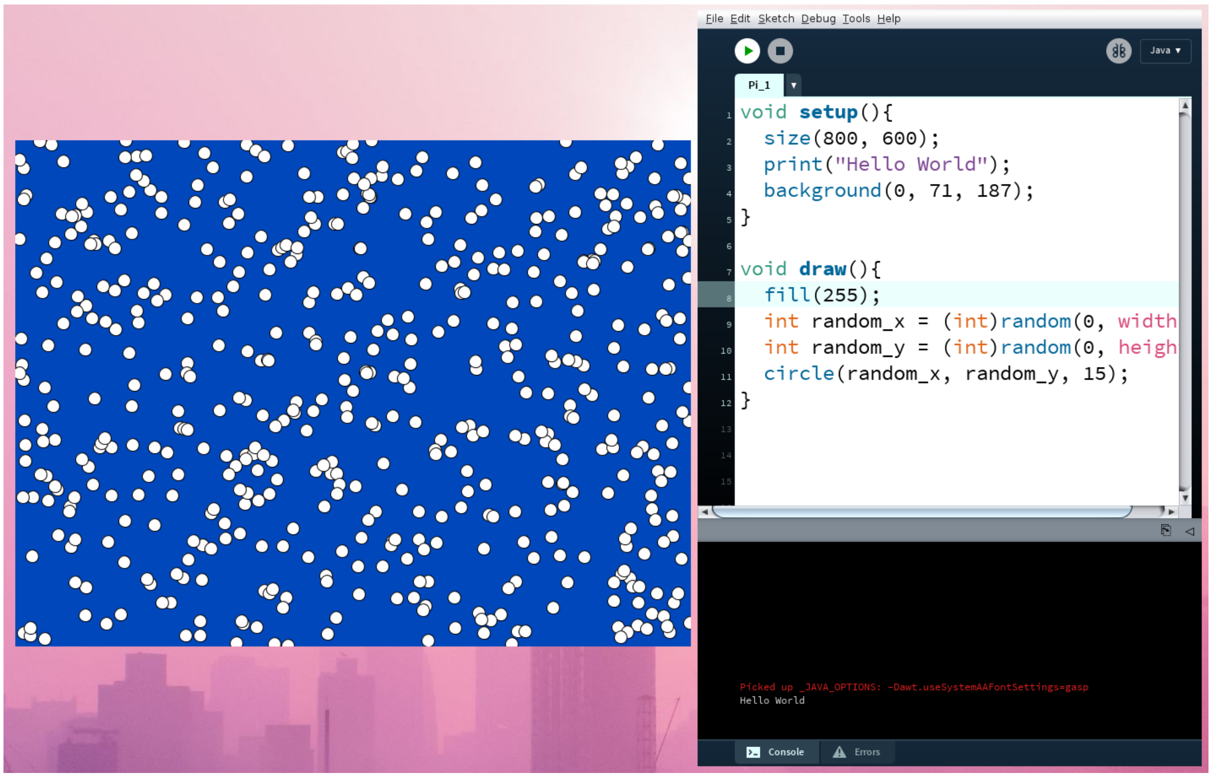
Draw

Another useful built-in function Processing provides, similar to setup, is called draw. However, instead of just being called once at the beginning of our program, it is called every time we want to render something onto the screen.
Depending on the type of monitor you're using, this could be in the region of 60 times a second, up to 144 times a second.

Add the following to your sketch, this will be after ALL the code we have written so far:

```
void draw(){
  fill(255);
  int random_x = (int)random(0, width);
  int random_y = (int)random(0, height);
  circle(random_x, random_y, 15);
}
```

When running the sketch you should now have circles being painted in rapidly.



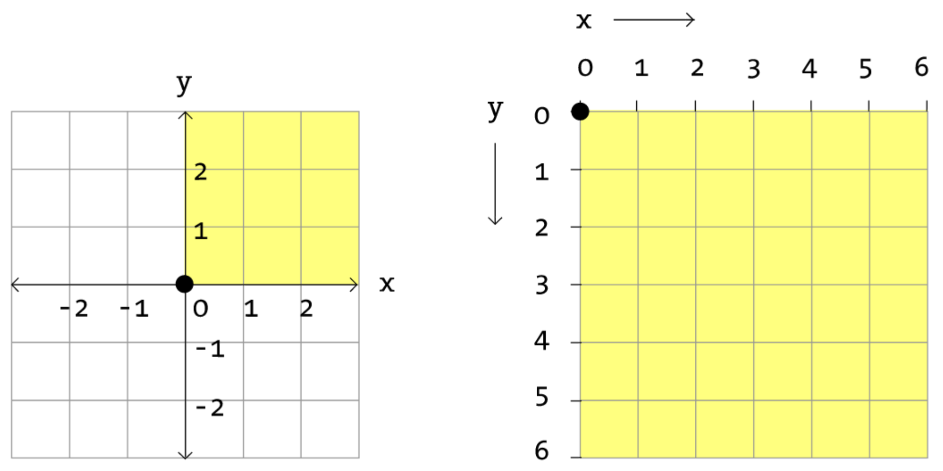
```
void setup(){
  size(800, 600);
  print("Hello World");
  background(0, 71, 187);
}

void draw(){
  fill(255);
  int random_x = (int)random(0, width
  int random_y = (int)random(0, heigh
  circle(random_x, random_y, 15);
}
```

```
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=gasp
Hello World
```

Here we define x, and y coordinates. These are the same concepts from graphing which you may have come across in mathematics.
X goes from left to right, from 0 to the width (in our case 800). Y goes from top to bottom (This is backwards from what Mathematics usually does!) from 0 to the height (in our case 600).

Using this coordinate system, the position (0,0) - would equal the top left corner. Likewise (10, 50) would be 10 along the x-axis (left-right), and 50 down the y-axis (top-bottom).

See the image below. Left is how we think of graphing in mathematics. Right is how Computer Scientists see things! We start counting at 0 too, think of this as the origin.



Let’s break down that code:

```
void draw(){
  fill(255);
  int random_x = (int)random(0, width);
  int random_y = (int)random(0, height);
  circle(random_x, random_y, 15);
}
```

Each time `draw()` is executed - Processing does this for us for these specially named blocks - the following should happen:

1. Set the `fill` colour to white (255).
2. Obtain a `random` x position between 0 (very left) and width (width of our window)
    a. Note, `(int)` is just saying our number is a counting number (0, 1, 2, 3) and we don’t want any decimal points.
3. Obtain a random y position between 0 (very top) and height (height of our window)
    a. `(int)random(..)` makes sure that we don’t have any 12.448 numbers. Pixels are discrete (I.e 0, 1, 2, 3), we cannot have .5 of a pixel displayed on your monitor!
4. Using the previously obtained random x, y position for our screen, then create a circle of diameter 15, where the center of the circle is at (x,y).

Just like in mathematical algebra, we can assign variables to be equal to some expression; ie. it can give us a value when we evaluate the expression, E.g. y = 5x + 4.
At the moment in time that we execute y = 5x + 4. We look up the value of x (E.g 2), multiply by 5, then add 4 to the result. That value is then stored in y. This would make y = 10 + 4, which is 14.

<u>Sub-Task</u> - Try the following:

1. Write the line: `int x = 2;`
2. After this line, write a line of code which makes `int y = 5x + 4;`
3. Change your Hello World line to print the value of x. Before we passed print() a ‘string’ (“ “ marks), this time we’re just going to give it the variable x between the print(and).
4. Now replace that line, and print the value of y. Hopefully, these should align with the example provided above (y = 14, and x = 2).

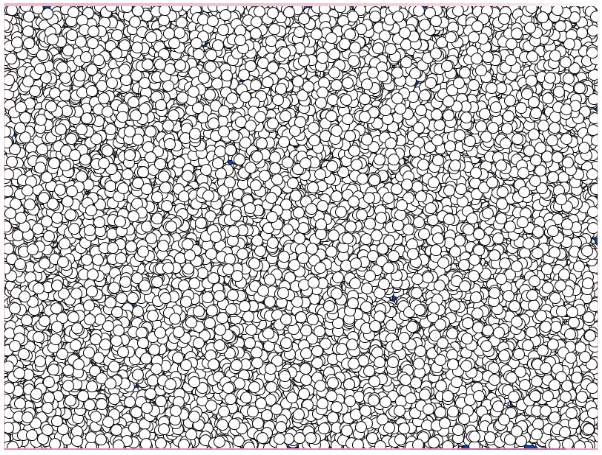
Back to our main task, we wanted a variable called ‘random_x’ to be a number between 0 and the width of our canvas.

We can then use ‘random_x’ afterwards, just like we do with algebra, to use the value stored in it. To continue our mathematical analogy, that’s like me then saying that z = 5y which is just 5 multiplied by whatever y equates to.

UNIVERSITY
OF HULL

UNIVERSITY
OF HULL

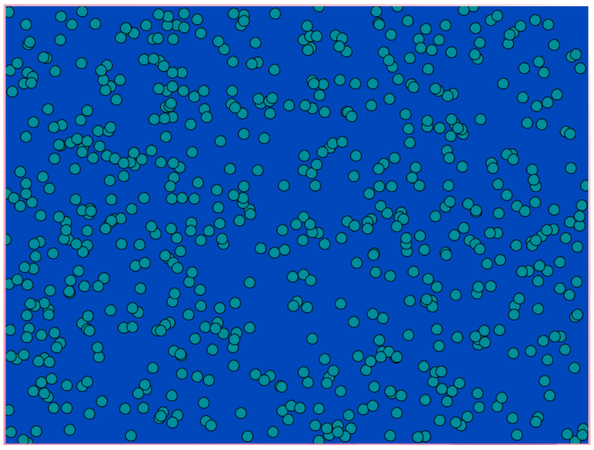
As our draw function is called each time we want to render a frame (multiple times a second!), we eventually end up with our entire page being painted as shown below.



Task: Can you change the circles so they are filled in with the colour RGB R:0 G:143 B:157 instead of white?

Hint: `fill()` works exactly like `background()`. What did we do with the numbers given to the background function to make it RGB colours?

Your output should now look similar to the following:



Task 2: Can you print the random_x, and random_y coordinates to the console after we have drawn the circle? We should be able to give the print function our variable. It will then look up what the value is associated with random_x.
We can also use something called println which puts things on a new line for us.

E.g println(random_x);

Note: This will fill the console up a lot! Multiple times per second.

Experiment with the concepts we've looked at in this worksheet. Can you change the colours? What about making the circles only appear in part of the screen, rather than the entire screen? Could you use random with some colour values to make rainbow circles?