Inheritance Lab 1

Lab 1 - Setting Up Processing

info

Open the File

You have to tell Processing the file you want to open. In the Processing window, click `File` then `Open...`. On the left towards the bottom of the list, click `workspace`. Double click on `code`, double click on `inheritance`, and double click on `InheritanceLab`. Finally, open the `InheritanceLab.pde` file. This file will be used for the animation.

This lab is going to use Processing to create some animations using inheritance. We are going to have a window that is 400 pixels by 400 pixels with a black background.

```
void setup() {
  size(400, 400);
}

void draw() {
  background(0);
}
```

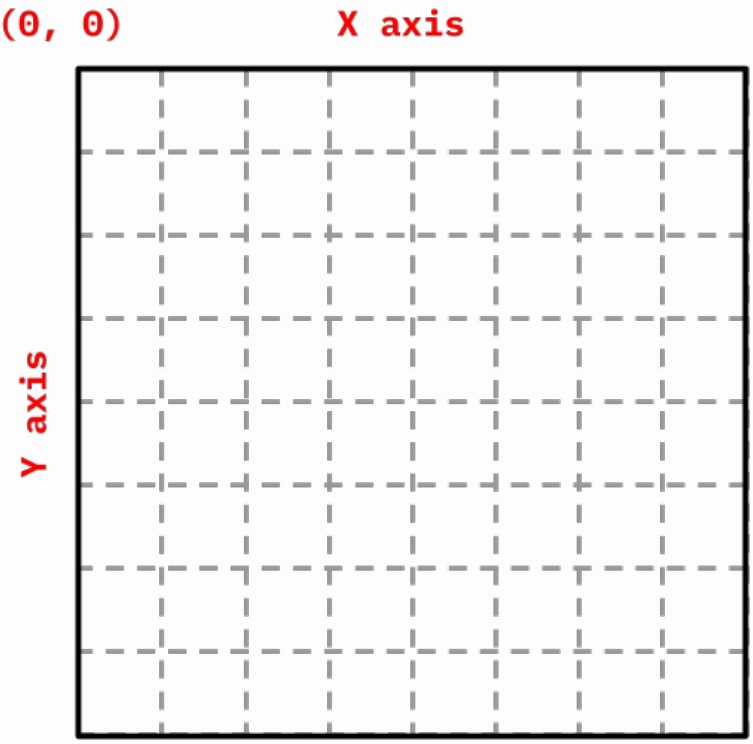
info

Try It

Click the triangle button to run you program. You should see a 400 x 400 window appear. Close the window or click the square button to stop the program. **Note**, you need to manually save your work in Processing. Click `File` and then `Save`, or press `Ctrl + S` on the keyboard to save your work.

Processing Refresher

- The Window - Pygame programs are run in a window, which is a grid of x- and y-coordinates. The origin point (0, 0) is located in the top-left corner. Use window coordinates to position objects in the window.



- Colors - Processing uses the RGB (red, green, blue) system of defining colors. Color values range between 0 and 255, and these values are mixed together. Use this website to help you find the RGB value you want. The `stroke` command colors the line for shapes and `fill` fills the shape with a color.
- This lab will use the `beginShape()` and `endShape(CLOSE)` commands. These commands tell Processing that you are going to draw a series of vertices and Processing will connect them (in order) with lines. The `CLOSE` command tells Processing to draw a line between the last vertex and the first vertex.

```
void draw() {
  noStroke();
  fill(255, 0, 0);
  background(0);
  beginShape();
  vertex(200, 100);
  vertex(300, 200);
  vertex(200, 300);
  vertex(100, 200);
  endShape(CLOSE);
}
```

challenge

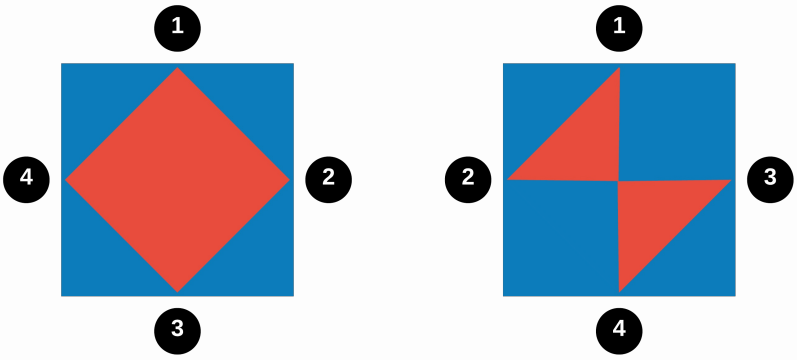
Try these variations:

- Change the order of vertices to be:

```
  vertex(200, 100);
  vertex(100, 200);
  vertex(300, 200);
  vertex(200, 300);
```

▼ **What happened?**

Processing draws a polygon in the order of the vertices. If you want a regular polygon, the vertices need to go in either clockwise or counterclockwise order. The first polygon used a clockwise pattern, while the second one used a zigzag pattern.



Inheritance Lab 2

Lab 2 - Creating the Parent Class

info

Open the File

You have to tell Processing the file you want to open. In the Processing window, click `File` then `Open...`. On the left towards the bottom of the list, click `workspace`. Double click on `code`, double click on `inheritance`, and double click on `InheritanceLab2`. Finally, open the `InheritanceLab2.pde` file. This file will be used for the animation.

This lab is all about inheritance, so we need to create a superclass from which we can inherit. We are going to create a class called `Hexagon` that draws a hexagon to window.

The `Hexagon` Class

Start the `Hexagon` declaration before the `setup` method. These attributes will aid in the drawing of the shape.

```
class Hexagon {
  private int xPosition;
  private int yPosition;
  private float radius;
  private int vertices;
  private float angle;
  private color clr;
  private float strokeW;
}
```

All of these attributes will need a getter method, but only the `angle` and `vertices` attributes need a setter method.

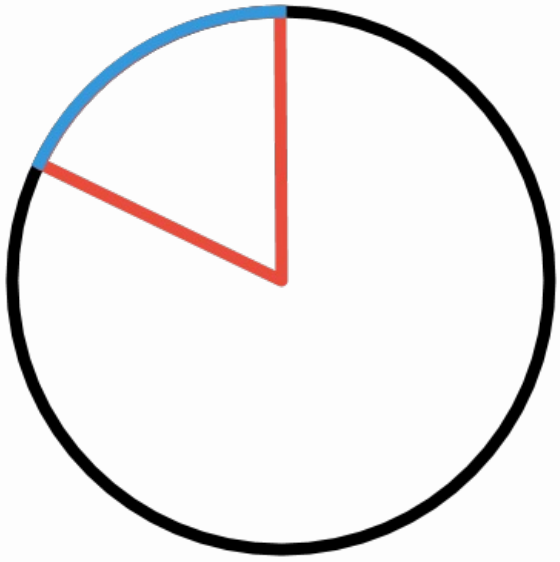
```
public float getRadius() {
  return radius;
}

public color getClr() {
  return clr;
}

public float getStrokeW() {
  return strokeW;
}

public int getVertices() {
  return vertices;
}

public void setVertices(int newVertices) {
  vertices = newVertices;
}

public float getAngle() {
  return angle;
}

public void setAngle(float newAngle) {
  angle = newAngle;
}

public int getXPosition() {
  return xPosition;
}

public int getYPosition() {
  return yPosition;
}
```

Processing will throw an error if the superclass does not have a default constructor. That is, a constructor without any parameters. Create a constructor that initializes each attribute with a starting value.

```
public Hexagon() {
  xPosition = 0;
  yPosition = 0;
  radius = 50;
  vertices = 6;
  angle = TWO_PI / vertices;
  clr = color(255, 255, 255);
  strokeW = 1;
}
```

▼ **Calculating the Angle Attribute**

The `angle` attribute is measured in radians. A radian is a unit of measure for angles. One radian is about 57.3 degrees. The are exactly 2π radians in any circle. To calculate the angle for each vertex, divide 2π by the number of vertices. In Processing, 2π is represented by `TWO_PI`. The image below shows that the a radian (blue) has the same length as the radius (red).



Now create a constructor that takes several parameters. There is no need to pass the constructor a value for `vertices`. A hexagon should always have six vertices, we can set that value manually. The same is true for `angle`. Once we know the number of vertices, we can calculate the `angle` attribute.

```
  public Hexagon(int x, int y, float r, color c, float sw) {
    xPosition = x;
    yPosition = y;
    radius = r;
    vertices = 6;
    angle = TWO_PI / vertices;
    clr = c;
    strokeW = sw;
  }
```

Test Your Code

Before moving on, we need to test the `Hexagon` object to make sure both constructors work as expected. After the `Hexagon` class and before the `setup` method, create two `Hexagon` objects. In the `setup` method, initialize each object, one with the default constructor and the other with some arguments. Processing will create a dark gray window, but it will not draw anything to the screen.

```
Hexagon h1;
Hexagon h2;

void setup() {
  size(400, 400);
  h1 = new Hexagon();
  h2 = new Hexagon(200, 200, 200, color(255, 0, 0), 3);
}

void draw() {
  background(55);
}
```

important

Save Your Work

If Processing runs your code without any errors, be sure to save your work. This **will not** happen automatically. Click `File` and then `Save`, or press `Ctrl + S` on the keyboard to save your work.

▼ **Code**

```
  class Hexagon {
```

```
  private int xPosition;
  private int yPosition;
  private float radius;
  private int vertices;
  private float angle;
  private color clr;
  private float strokeW;

  public float getRadius() {
    return radius;
  }

  public color getClr() {
    return clr;
  }

  public float getStrokeW() {
    return strokeW;
  }

  public int getVertices() {
    return vertices;
  }

  public void setVertices(int newVertices) {
    vertices = newVertices;
  }

  public float getAngle() {
    return angle;
  }

  public void setAngle(float newAngle) {
    angle = newAngle;
  }

  public int getXPosition() {
    return xPosition;
  }

  public int getYPosition() {
    return yPosition;
  }

  public Hexagon() {
    xPosition = 0;
    yPosition = 0;
    radius = 50;
```

```
    vertices = 6;
    angle = TWO_PI / vertices;
    clr = color(255, 255, 255);
    strokeW = 1;
  }

  public Hexagon(int x, int y, float r, color c, float sw) {
    xPosition = x;
    yPosition = y;
    radius = r;
    vertices = 6;
    angle = TWO_PI / vertices;
    clr = c;
    strokeW = sw;
  }
}

Hexagon h1;
Hexagon h2;

void setup() {
  size(400, 400);
  h1 = new Hexagon();
  h2 = new Hexagon(200, 200, 200, color(255, 0, 0), 3);
}

void draw() {
  background(55);
}
```

Inheritance Lab 3

Lab 3 - Drawing the Hexagon Object

info

Open the File

You have to tell Processing the file you want to open. In the Processing window, click `File` then `Open...`. On the left towards the bottom of the list, click `workspace`. Double click on `code`, double click on `inheritance`, and double click on `InheritanceLab2`. Finally, open the `InheritanceLab2.pde` file. This file will be used for the animation.

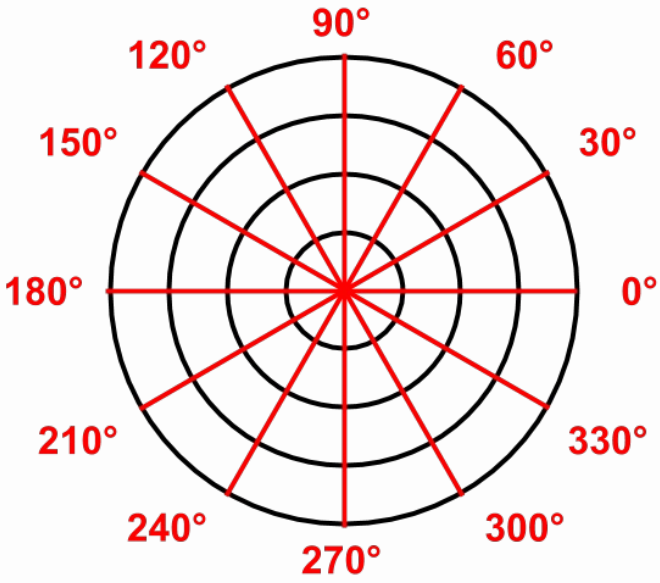
Create a method called `show` in the `Hexagon` class. We are going to use `beginShape` and `endShape` to draw the hexagon. Tell Processing not to fill the shape. Instead set the `stroke` (color of the line) and `strokeWeight` (line thickness). Since there are six vertices in a hexagon, create a for loop that runs six time. Each iteration will draw a vertex a given length from the center of the shape. Create the the loop but do not put anything inside of it just yet.

```
public void show() {
  beginShape();
  noFill();
  stroke(clr);
  strokeWeight(strokeW);
  for (int i = 0; i < vertices; i++) {

  }
  endShape(CLOSE);
}
```

Polar to Cartesian Transformation

The `Hexagon` class is designed to position the shape according to `xPosition` and `yPosition`. The six vertices radiate out from the center according to the `radius` attribute, and the spacing between each vertex is controlled by `angle`. Using angles and a distance from a central point to calculate a

position is referred to as the polar coordinate system. The red lines show different angles and the black concentric circles represent a distance from the center. The intersection of the red line and black circle is a location.



Polar Coordinates

The problem is that the Processing window uses the Cartesian coordinate system. With a little bit of trigonometry, you can convert the polar coordinate system to the Cartesian coordinate system. The x- and y-coordinates for each point of the hexagon can be calculated with the following formulas.

```
x = cosine(angle) * radius
y = sine(angle) * radius
```

These formulas need to be tweaked a bit for the `Hexagon` class. First, the above formulas assume you are setting a position relative to the origin point (the top-left corner of the window). Instead, we want to position each vertex in relation to the center of the hexagon. So offset the x-coordinate of the vertex with the x-position of the center of the hexagon. Similarly, offset the y-coordinate of the vertex with the y-position of the center of the hexagon. Since there are six vertices, use a for loop to calculate each vertex. Use local variables to calculate the x- and y-values and then draw a vertex.

```
  public void show() {
    beginShape();
    noFill();
    stroke(clr);
    strokeWeight(strokeW);
    for (int i = 0; i < vertices; i++) {
      float x = cos(angle * i) * radius + xPosition;
      float y = sin(angle * i) * radius + yPosition;
      vertex(x, y);
    }
    endShape(CLOSE);
  }
```

Test Your Code

To be sure that the `Hexagon` class works, we are going to add an object and draw it to the window. Before the `setup` method create a `Hexagon` object. Inside the `setup` method instantiate the object. We want it to be centered in the window, so set the x-position to `width/2` and the y-position to `height/2`. Give it a radius of `125`, a color of `(179, 55, 113)`, and a stroke weight of `3`. Finally go into the `draw` method and call the `show` method on the `Hexagon` object.

```
Hexagon h;

void setup() {
  size(400, 400);
  h = new Hexagon(200, 200, 125, color(179, 55, 113), 3);
}

void draw() {
  background(55);
  h.show();
}
```

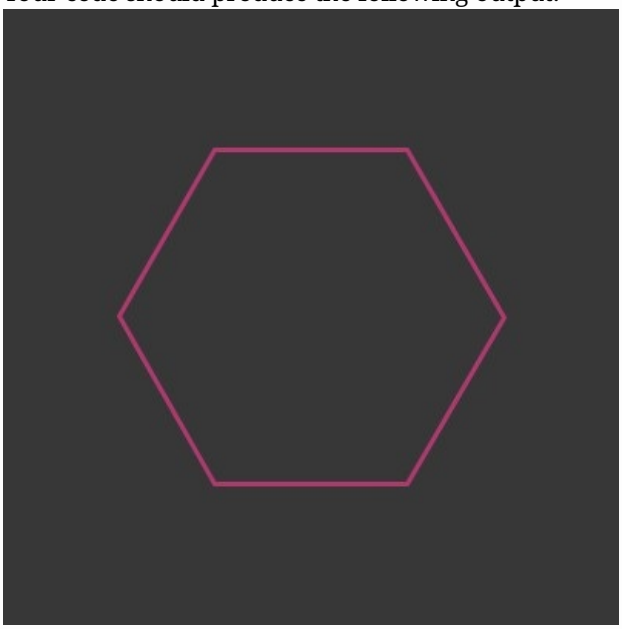
important

Save Your Work

If Processing runs your code without any errors, be sure to save your work. This **will not** happen automatically. Click `File` and then `Save`, or press `Ctrl + S` on the keyboard to save your work.

▼ **Code**

Your code should produce the following output:



```
class Hexagon {
  private int xPosition;
  private int yPosition;
  private float radius;
  private int vertices;
  private float angle;
  private color clr;
  private float strokeW;

  public float getRadius() {
    return radius;
  }

  public color getClr() {
    return clr;
  }

  public float getStrokeW() {
    return strokeW;
  }

  public int getVertices() {
    return vertices;
  }

  public void setVertices(int newVertices) {
    vertices = newVertices;
  }
```

```
public float getAngle() {
  return angle;
}

public void setAngle(float newAngle) {
  angle = newAngle;
}

public int getXPosition() {
  return xPosition;
}

public int getYPosition() {
  return yPosition;
}

public Hexagon() {
  xPosition = 0;
  yPosition = 0;
  radius = 50;
  vertices = 6;
  angle = TWO_PI / vertices;
  clr = color(255, 255, 255);
  strokeW = 1;
}

public Hexagon(int x, int y, float r, color c, float sw) {
  xPosition = x;
  yPosition = y;
  radius = r;
  vertices = 6;
  angle = TWO_PI / vertices;
  clr = c;
  strokeW = sw;
}

public void show() {
  beginShape();
  noFill();
  stroke(clr);
  strokeWeight(strokeW);
  for (int i = 0; i < vertices; i++) {
    float x = cos(angle * i) * radius + xPosition;
    float y = sin(angle * i) * radius + yPosition;
    vertex(x, y);
  }
  endShape(CLOSE);
```

```
  }
}

Hexagon h;

void setup() {
  size(400, 400);
  h = new Hexagon(200, 200, 125, color(179, 55, 113), 3);
}

void draw() {
  background(55);
  h.show();
}
```

challenge

Try these variations:

- Instantiate the `Hexagon` object with the default constructor (no parameters). You should see part of a white hexagon in the top-left corner of the window.

▼ **Solution**

```
Hexagon h;

void setup() {
  size(400, 400);
  h = new Hexagon();
}

void draw() {
  background(55);
  h.show();
}
```

- Add more `Hexagon` objects to the window. Name them `hexagon2`, `hexagon3`, etc. and draw them to the window.

▼ **Solution**

Here is one possible solution:

```
Hexagon h, h1, h2, h3;

void setup() {
  size(400, 400);
  h = new Hexagon(300, 95, 45, color(12, 231, 88), 7);
  h1 = new Hexagon(104, 311, 115, color(102, 31, 248), 1);
  h2 = new Hexagon(184, 50, 205, color(192, 131, 221),
      10);
  h3 = new Hexagon(348, 211, 215, color(1, 131, 255), 25);
}

void draw() {
  background(55);
  h.show();
  h1.show();
  h2.show();
  h3.show();
}
```

Inheritance Lab 4

Lab 4 - Extending the Superclass

info

Open the File

You have to tell Processing the file you want to open. In the Processing window, click `File` then `Open...`. On the left towards the bottom of the list, click `workspace`. Double click on `code`, double click on `inheritance`, and double click on `InheritanceLab2`. Finally, open the `InheritanceLab2.pde` file. This file will be used for the animation.

Now that we have a superclass, it is time to extend it through inheritance. We an `AnimatedHexagon` class that grows and shrinks. Start by creating the `AnimatedHexagon` class as a subclass of the `Hexagon` class. Then create the constructor and pass the arguments to the constructor of the superclass.

```
class AnimatedHexagon extends Hexagon {

    public AnimatedHexagon(int x, int y, float r, color c, float
        sw) {
      super(x, y, r, c, sw);
    }
}
```

Next, replace references to `Hexagon` with `AnimatedHexagon`. In the main loop, call the `show` method for the `AnimatedHexagon` object. Run your code to make sure you see a hexagon.

```
AnimatedHexagon h;

void setup() {
  size(400, 400);
  h = new AnimatedHexagon(width/2, height/2, 175, color(39, 209,
        220), 3);
}

void draw() {
  background(0);
  h.show();
}
```

Animating the Hexagon

We are going to animate the hexagon by overriding the `show` method. Instead of having a fixed radius (the distance from the vertices from the center of the hexagon), we are going to create a radius that increases and decreases over time. This will give the appearance that the hexagon is growing and shrinking.

Start by adding the `time` attribute to the `AnimatedHexagon` constructor. As this value changes, the shape will animate. This value needs to be a `float` since we want to increment this attribute by amounts less than 1.

```
class AnimatedHexagon extends Hexagon {
  private float time;

  public AnimatedHexagon(int x, int y, float r, color c, float
        sw) {
    super(x, y, r, c, sw);
    time = 0;
  }
}
```

Override the `show` method. There are three small differences in this new method. The local variable `length` is calculated with the helper method `calculateLength`. This variable will replace the `radius` attribute. Be sure to use `length` instead of `radius` when calculating `x` and `y`.

```
public void show() {
  beginShape();
  noFill();
  stroke(getClr());
  strokeWeight(getStrokeW());
  float length = calculateLength();
  for (int i = 0; i < getVertices(); i++) {
    float x = cos(getAngle() * i) * length + getXPosition();
    float y = sin(getAngle() * i) * length + getYPosition();
    vertex(x, y);
  }
  endShape(CLOSE);
```

Now we need to define `calculateLength`, which will return a float. Increment the `time` attribute by `.0.01`. This amount represents the speed of the animation. Increase the amount to increase the speed, decrease it to slow down the animation. The variable `length` represents the new distance between the vertex and the center of the hexagon. `sin` is used because this function will return a value between -1 and 1. When the result is positive, the hexagon grows. When the result is negative, the hexagon shrinks. This will create a never-ending animation. It is important to multiply `sin(time)` by `getRadius()`. If not, maximum size the hexagon can have is 1 pixel. Return `length` so it can be used to calculate the coordinate pairs for each vertex.

```
public float calculateLength() {
  time += 0.01;
  float length = sin(time) * getRadius();
  return length;
}
```

Test Your Code

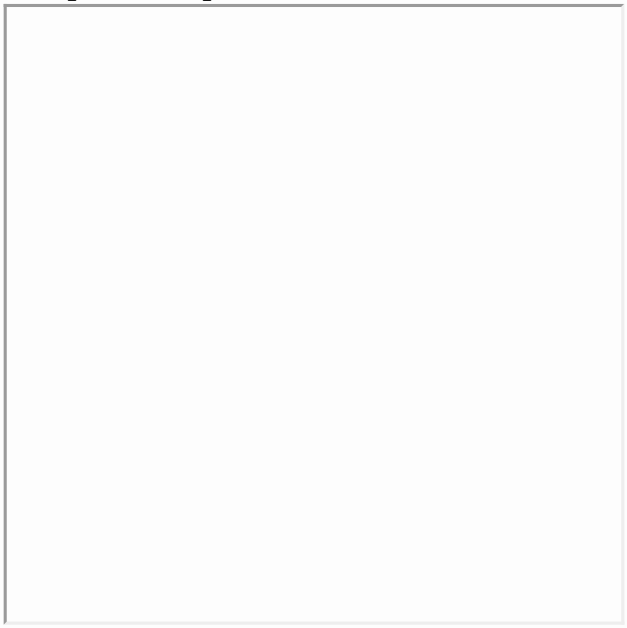
Your octagon should now grow and shrink. Adjust the animation by changing attributes like `time`, `radius`, and `clr`.

important

Save Your Work

If Processing runs your code without any errors, be sure to save your work. This **will not** happen automatically. Click `File` and then `Save`, or press `Ctrl + S` on the keyboard to save your work.

▼ **Expected Output**



Inheritance Lab Challenge

Problem

In the IDE to the left, the class `MP3` is already defined. Use this class to do the following things:
* Create the class `Podcast` that inherits from `MP3`
* Override the constructor such that the podcast has the following attributes:
* `name` - a string that is the name of the podcast
* `title` - a string that is the title of the episode
* `length` - an integer that has the length of the podcast **in seconds**
* `genre` - a string that is the genre of the podcast
* `date` - a string that represents when the podcast was released to the public

Expected Output

Declare an instance of the `Podcast` class as shown below.

```
//add code below this line

Podcast p = new Podcast("Planet Money", "Hollywood's Black
    List", 1460, "economics", "10 July 2020");


//add code above this line
```

The table below shows the method calls (left) and the output (right). Your class will be expected to work with all of these method calls.

Method Call	Return Value
`System.out.println(p.displayName());`	`The name is Planet Money`
`System.out.println(p.displayTitle());`	`The title is Hollywood's Black List`
`System.out.println(p.displayLength());`	`The length is 24 minutes and 20 seconds`
`System.out.println(p.displayGenre());`	`The genre is economics`
`System.out.println(p.displayDate());`	`The date is 10 July 2020`