Target practice

Use Python to draw a target and score points by hitting it with arrows

Step 1 You will make

Use Python, with the `p5` graphics library, to draw a target and score points by hitting it with arrows.

You will:

- Personalise your game with **RGB colours**
- Use **conditional statements** (`if`, `elif`, `else`) to make decisions
- Position shapes with **x, y coordinates**

> **RGB colours** have values between 0 and 255 for each of R(ed), G(reen), and B(lue). Purple has values R,G,B (128, 0, 128) — medium amounts of red and blue, with no green. Video game artists and graphic designers work with RGB colours.

> The oldest evidence of **archery** comes from the Sibudu Cave in KwaZulu-Natal, South Africa. Remains of stone and bone arrowheads have been found, which date to between 60,000 and 70,000 years ago.

Step 2 Create a background

The sky and grass are made by writing code to draw coloured rectangles.

Open the **Archery starter** (https://trinket.io/python/9973649e5c) project.

If you have a Trinket account, you can click on the **Remix** button to save a copy to your `My Trinkets` library.

The starter project has some code already written for you to import the `p5` library, you will use this library to build your archery game.

p5 Processing library

The p5 library allows you to create drawings, animations and simple games.

main.py

```
4 from p5 import *
```

> The **p5 library** provides graphic functions for drawing, animation and data visualisation. Artists, animators and designers use the p5 Processing library for creative coding.

There are two functions that every project using `p5` needs to **define**:

- **setup()** - runs once when the program starts to set properties like screen size
- **draw()** - runs repeatedly and defines what will be sketched

You also need to **call** the `run()` function:

- **run()** - starts the p5 project by calling the `setup()` function followed by repeatedly calling the `draw()` function

The `fill()` function sets the inside colour of shapes. The starter project already contains some RGB colours you can use to do this.

Find your `draw()` function and prepare to draw the sky by adding indented code to set the `fill()` colour to `sky`:

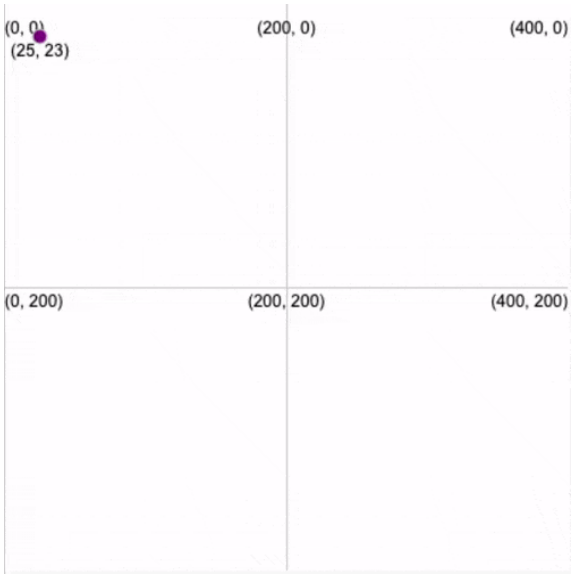
main.py — draw()

```
18  def draw():
19    #Things to do in every frame
20    sky = color(92, 204, 206) #Red = 92, Green = 204, Blue = 206
21    grass = color(149, 212, 122)
22    wood = color(145, 96, 51)
23    outer = color(0, 120, 180)
24
25    fill(sky)
```

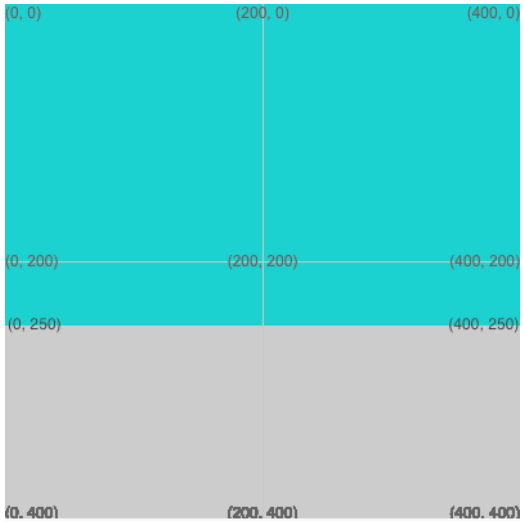
The `size()` function call in `setup()` sets the screen size to 400 pixels by 400 pixels.

Coordinates in p5

Coordinates in p5 start from an **origin point (0,0)** in the top-left of the screen. This top-left positioning of x = 0 and y = 0 is commonly used when programming apps and games. If you have used Scratch or plotted charts on paper you might be used to seeing x = 0 and y = 0 in the centre.



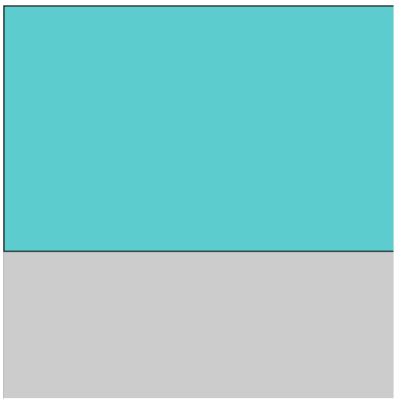
After your `fill()` code, draw a `rect()` for the sky with top-left coordinates (`0`,`0`), a width of `400` to match the width of the screen and a height of `250`.



main.py — draw()

```
25 fill(sky)
26 rect(0, 0, 400, 250) #Start x, start y, width, height
```

Test: Run your code to see the sky you've drawn. Remember that with the `p5` library, the `run()` function calls the `setup()` function once, then the `draw()` function repeatedly.



That's a bit strange: there's a black line around your sky! This is because, when the program starts, it automatically sets a black border — called a **stroke** — around everything it draws.

Turn off the stroke by adding `no_stroke()` before you start drawing the sky.

main.py — draw()

```
23 outer = color(0, 120, 180)
24
25 no_stroke()
26 fill(sky)
27 rect(0, 0, 400, 250) #x, y, width, height
```

Test: Run your project again to check that the stroke has gone.

`fill()` changes the fill colour for all shapes drawn until `fill()` is called again with a new colour.

Change the `fill()` colour to `grass` and add another `rect(x, y, width, height)`.

This rectangle needs to be positioned below the sky at coordinates (0, 250), so that it starts in the lower part of the screen.

main.py — draw()

```
23  outer = color(0, 120, 180)
24
25  no_stroke()
26  fill(sky)
27  rect(0, 0, 400, 250) #x, y, width, height
28  fill(grass)
29  rect(0, 250, 400, 150)
```

Test: Run your project again to view the finished background.

Save your project

Step 3 Draw your target

The target stand is a triangle shape. The target is made with coloured circles — smaller circles are worth more points than larger ones.

Shapes are drawn in the order that the lines of code run. The triangular wooden stand is partly behind the target circles so it must be drawn first.

Imagine cutting all the shapes out of paper. Depending on how you arrange and overlap that paper, the final result could look very different.

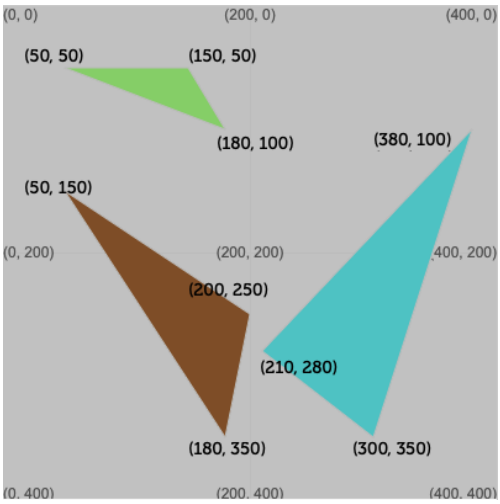
Draw the stand

When you call the `triangle()` function, you need to provide three sets of coordinates, `x1, y1, x2, y2, x3, y3` each representing the position of one of the triangle's corners.

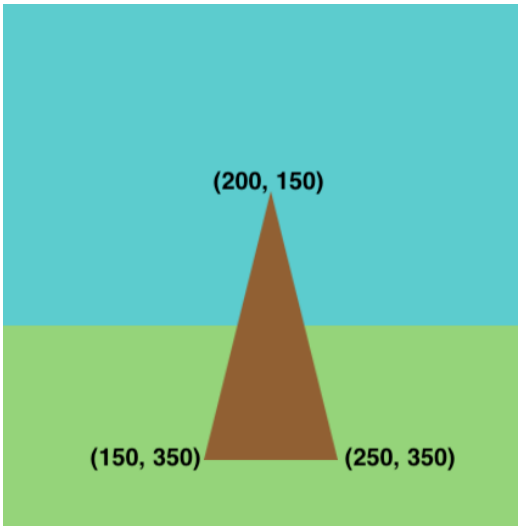
Triangle coordinates

Here are three example triangles, each with different sets of coordinates. Look at the grid position of each to see how the `x` and `y` coordinates position the corners of the triangles:

- Green triangle: triangle(50, 50, 150, 50, 180, 100)
- Blue triangle: triangle(210, 280, 300, 350, 380, 100)
- Brown triangle: triangle(50, 150, 200, 250, 180, 350)



Draw a `triangle()` for the stand with corners at (150, 350), (200, 150), and (250, 350).



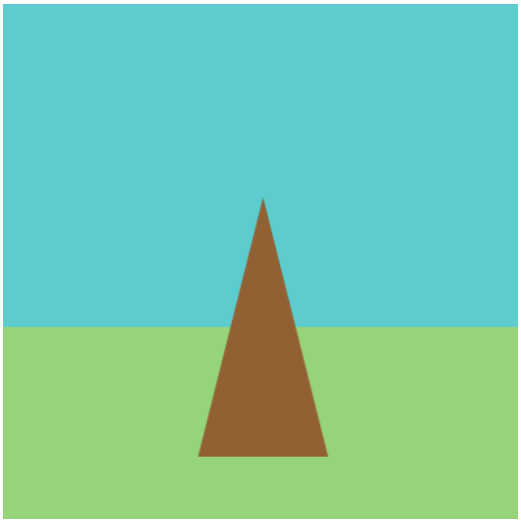
main.py - draw()

```
fill(grass)
rect(0, 250, 400, 150) #x, y, width, height

fill(wood) #Set the stand fill colour to brown
triangle(150, 350, 200, 150, 250, 350)
```

Tip: We have added comments to our code, like `#Set the stand fill colour to brown`, to tell you what it does. You don't need to add these comments to your code, but they can be helpful to remind you what lines of code do.

Test: Run your code to see the stand for your target.



Draw the target

The largest part of the target will be a blue **circle** made by using the `ellipse()` function. An ellipse is a shape with a single side and no corners. It can be squashed, like an oval, or perfecly round, like a circle.

An ellipse needs `x` and `y` coordinates, width, and height. The `x` and `y` coordinates of an ellipse are the centre position.

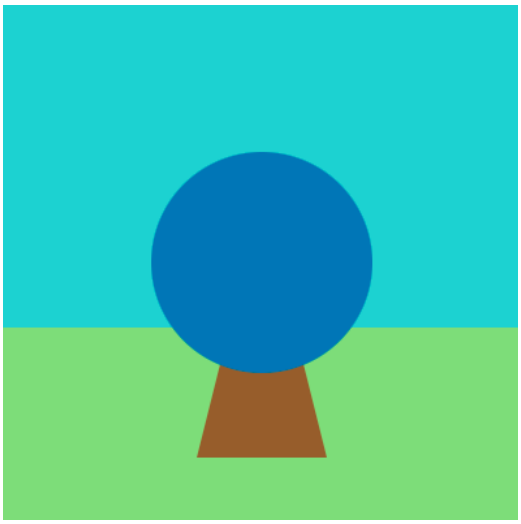
The blue circle will cover the brown triangle where they overlap, because the circle was drawn later.

Tip: To make a circle, the **width** and **height** must be the same.

main.py - draw()

```
fill(wood)
triangle(150, 350, 200, 150, 250, 350)
fill(outer)
ellipse(200, 200, 170, 170) #Outer circle. 200, 200 is the middle of the screen
```

Test: Run your code to see the first large blue circle.



Create two new variables to store colours `inner` and `bullseye` for the remaining circles.

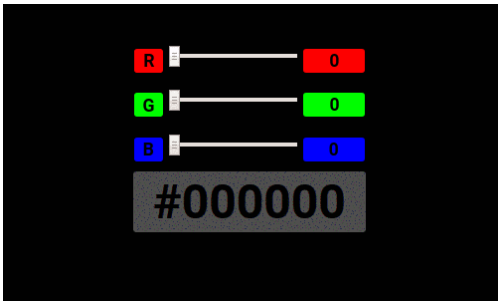
Assign colours to the `inner` and `bullseye` variables using `color()`.

The `color()` function expects three numbers: one each for red, green, and blue.

We used numbers that give traditional archery target colours, but you can use whatever colours you like as long as they are different from each other.

RGB colours

When we want to represent a colour in a computer program, we can do this by defining the amounts of red, blue, and green that are combined to make up that colour. These amounts are stored as a number between 0 and 255.



Here's a table showing some colour values:

Red	Green	Blue	Colour
255	0	0	Red
0	255	0	Green
0	0	255	Blue
255	255	0	Yellow
255	0	255	Magenta
0	255	255	Cyan

You can find a nice **colour picker to play with at w3schools** (https://www.w3schools.com/colors/colors_rgb.asp).

main.py - draw()

```
18  def draw():
19    #Things to do in every frame
20
21    sky = color(92, 204, 206)
22    grass = color(149, 212, 122)
23    wood = color(145, 96, 51)
24    outer = color(0, 120, 180) #Blue
25    inner = color(210, 60, 60) # Red
26    bullseye = color(220, 200, 0) #Yellow
```

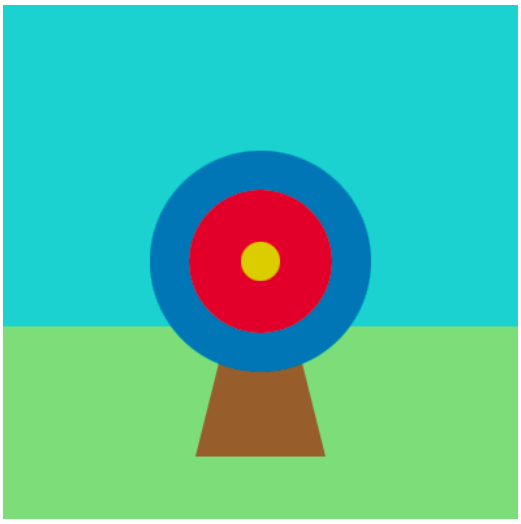
The target is made of different-sized circles with the same centre coordinates (200, 200) — the middle of the screen.

Add two more circles to represent an inner circle and the bullseye. Change the `fill()` before drawing each circle.

main.py - draw()

```
fill(wood)
triangle(150, 350, 200, 150, 250, 350) #Stand
fill(outer)
ellipse(200, 200, 170, 170) #Outer circle
fill(inner)
ellipse(200, 200, 110, 110) #Inner circle
fill(bullseye)
ellipse(200, 200, 30, 30) #Bullseye
```

Test: Run your project again to see the target with three coloured circles. Change the colours until you are happy with them.



Debug: Python uses the American spelling of 'color' (without a 'u') so make sure you do the same.

Save your project

Step 4 Fire your arrow

Now it's time to add an arrow that moves randomly across the target area.

Find the comment **#The shoot_arrow function goes here** and below it add code to define your `shoot_arrow()` function.

Add a small `ellipse()` in the centre of the screen to represent the arrow.

main.py — shoot_arrow()

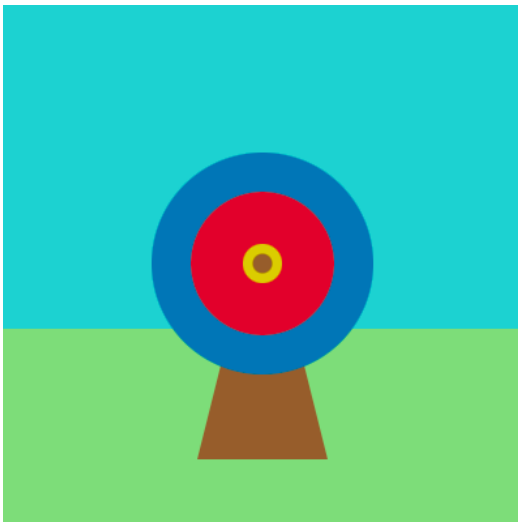
```
10 #The shoot_arrow function goes here
11 def shoot_arrow():
12   ellipse(200, 200, 15, 15)
```

Go to the `draw()` code that creates the target and add code at the end to set the `fill()` to `wood`, then call your new `shoot_arrow()` function.

main.py — draw()

```
41 fill(bullseye)
42 ellipse(200, 200, 30, 30)
43
44 fill(wood)
45 shoot_arrow()
```

Test: Run you code and see the arrow appear in the bullseye.



Computer games, videos, and animations create the effect of movement by showing lots of images one after another. Each image is called a **frame** . The speed that the image changes at is called the frame rate and is given in fps or frames per second.

The `frame_rate(2)` line in `setup()` sets the frame rate to 2 frames per second.

The `draw()` function is called every frame. You are going to draw the arrow in a random position each time `draw()` is called.

The background and target will be drawn over the old arrow. This means you only see one arrow at a time.

Find the `import` statements, at the top of your code, you are going to use `randint` from the `random` library.

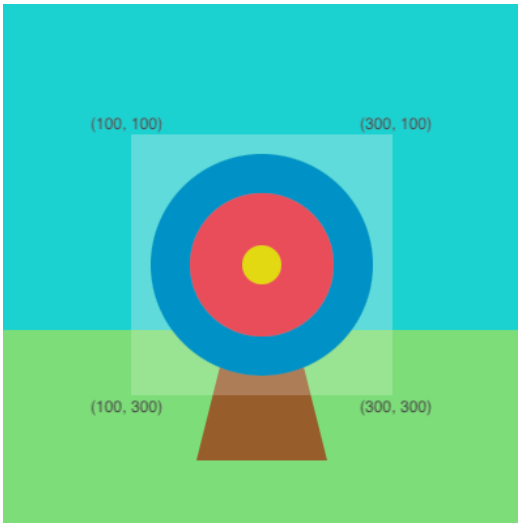
main.py

```
#Import library code
from p5 import *
from math import *
from random import randint
```

Go to your `shoot_arrow()` function and add two new `arrow_x` and `arrow_y` variables to store random numbers between `100` and `300`.

This will let some shots miss the target, without them going all the way to the edges of your game.

Change your `ellipse()` to use the new variables to position your arrow.



main.py — shoot_arrow()

```
#The shoot_arrow function goes here
def shoot_arrow():
  arrow_x = randint(100, 300)
  arrow_y = randint(100, 300)
  ellipse(arrow_x, arrow_y, 15, 15) #Update to random coordinates
```

Get the colour the arrow hits

The `get()` function returns the colour of a pixel.

> A **pixel**, short for picture element, is a single coloured dot within an image. Images are made up of lots of coloured pixels.

You need to store the colour that the arrow is aiming at before you draw an arrow on top of it.

Add code to store the `hit_color`. Use the `get()` function, to get the colour of the pixel at the `arrow_x` and `arrow_y` coordinates — the centre of the arrow.

main.py — shoot_arrow()

```
#The shoot_arrow function goes here
def shoot_arrow():
  arrow_x = randint(100, 300)
  arrow_y = randint(100, 300)
  hit_color = get(arrow_x, arrow_y) #Save the colour before drawing the arrow
  ellipse(arrow_x, arrow_y, 15, 15)
```

Tip: The code to get the colour and save it needs to be **before** the code to draw the ellipse otherwise you will always save the wood colour of the arrow!

The `p5` library 'listens' for certain events, one of these is the press of the mouse button. When it detects that the button has been pressed, it will run whatever code it has been given in the `mouse_pressed()` function.

Find the comment **#The mouse_pressed function goes here** and below it add code to define your `mouse_pressed()` function.

Add code to print the amounts of red, green, and blue in the pixel the arrow lands on.

main.py - mouse_pressed()

```
#The mouse_pressed function goes here
def mouse_pressed():
  print( red(hit_color), green(hit_color), blue(hit_color) )
```

You have defined two functions `shoot_arrow()` and `mouse_pressed()`, both of these functions need to use the `hit_color` variable.

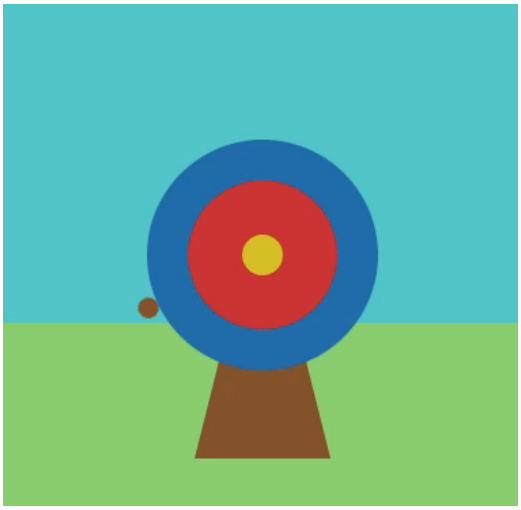
A variable that needs to be used thoughout a program is known as a **global variable**. Add code to your `shoot_arrow()` function to make `hit_color` a global variable:

main.py - shoot_arrow()

```
#The shoot_arrow function goes here
def shoot_arrow():
  global hit_color #Can be used in other functions
  arrow_x = randint(100, 300)
  arrow_y = randint(100, 300)
  hit_color = get(arrow_x, arrow_y) #Save the colour before drawing the arrow
  ellipse(arrow_x, arrow_y, 15, 15)
```

Test: Run your project. The arrow is redrawn at random coordinates.

The project gets the `hit_color` each time the arrow is redrawn and prints the colour value in the output area underneath the target.



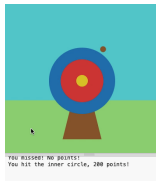
Debug: If you are seeing a message about `hit_colour` being 'not defined', then go back to `shoot_arrow()` and check that you have the `global hit_color` line.

Debug: Check the `print` line really carefully for commas and brackets.

Save your project

Step 5 Score points

In this step, you will add scores based on where the arrow hits.



The colour variables in the `draw()` function will be used to check the score in the `mouse_pressed()` function. To do this, they need to be set as global variables:

main.py

```
26 def draw():
27 #Things to do in every frame
28   global outer, inner, bullseye
29   sky = color(92, 204, 206) #Red = 92, Green = 204, Blue = 206
30   grass = color(149, 212, 122)
31   wood = color(145, 96, 51)
32   outer = color(0, 120, 180)
33   inner = color(210, 60, 60)
34   bullseye = color(220, 200, 0)
```

We use **conditions** all the time to make decisions. We could say 'if the pencil is blunt, then sharpen it'. Similarly, `if` conditions let us write code that does something different depending on whether a condition is true or false.

To `print` a message for the target's outer circle, add code to your `mouse_pressed()` function to check if the `hit_color` is `==` to `outer`.

Be careful when using the `=` symbol in Python:

- `=` is used for **assignment** — like `arrow_x = 200` to set the value of a variable
- `==` is used to test **equivalence** — like `hit_color == bullseye` — if the things on either side have the same value, then the test is `True`, otherwise it is `False`

Change the code in your `print()` to give a score:

main.py - mouse_pressed()

```
8  #The mouse_pressed function goes here
9  def mouse_pressed():
10   if hit_color == outer:
11     print('You hit the outer circle, 50 points!') #Like functions, 'if' statements are indented
```

Test: Run your project. Try to stop the arrow on the blue outer circle to see your message. The colour of the pixel at the centre of the arrow is the colour that is saved and checked.

Tip: `frame_rate()`, in `setup()`, controls how fast your game draws. If it's going too fast, set it to a lower number.

Debug: Make sure your code matches exactly and you indented the code inside your `if` statement. The indent tells Python that the code should only run if the condition is `True`.

As points will be scored if the arrow lands on the `inner` or `bullseye` circles too, `outer` is not the only circle you need to check. To do this, use `elif` (a shortened version of else - if).

> We use **else - if** to make decisions in real life. When you are painting a picture of the sky, you might check if there is a yellow paint for the sun. Else, if there is no yellow paint, you look for orange. Else, if there is no yellow or orange paint, you might use red — really lightly!

An `elif` can only be used with an `if` statement and, like an `if`, it checks a condition. If the condition is `True`, the `elif` runs some code.

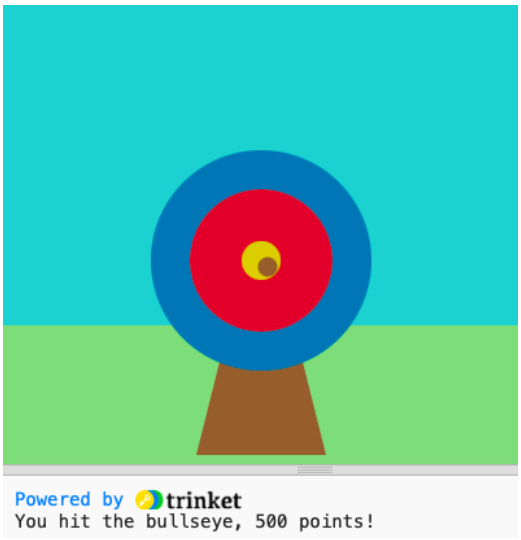
What makes `elif` different is that it will only make that check if the conditions of the `if` and any `elif`s before it are `False`.

Add `elif` statements for `inner` and `bullseye`.

main.py - mouse_pressed()

```
9  def mouse_pressed():
10   if hit_color == outer:
11     print('You hit the outer circle, 50 points!')
12   elif hit_color == inner:
13     print('You hit the inner circle, 200 points!')
14   elif hit_color == bullseye:
15     print('You hit the bullseye, 500 points!')
```

Test: Run your project. Try to stop the arrow on the red and yellow circles to see their messages.



Debug: Make sure your `elif` is at the same indentation level as your `if`, and the code inside your `elif` is at the same level as the code inside your `if`.

Debug: If you see a message about `inner` or `bullseye` being 'not defined', then go back to `draw()` and check that they are on the line that declares variables global.

```
global outer, inner, bullseye
```

There is one more decision you need to make: what happens if the arrow does not land on any of the target circles? To do this last check, you use `else`.

> We use **if … else** to make decisions. When you wake up, you check and if it is morning you get up, else you go back to sleep. Can you think of any if … else decisions you make?

Add code to `print` a message `else` none of the `if` and `elif` statements have been met.

main.py

```
9  def mouse_pressed():
10   if hit_color == outer:
11     print('You hit the outer circle, 50 points!')
12   elif hit_color == inner:
13     print('You hit the inner circle, 200 points!')
14   elif hit_color == bullseye:
15     print('You hit the bullseye, 500 points!')
16   else:
17     print('You missed! No points!')
```

Test: Run your project. Try to stop the arrow in the grass or sky to see the miss message. Change the number of points scored for the different colours if you like.

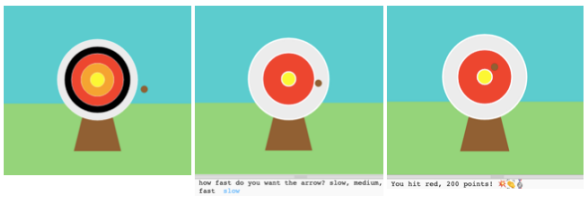
Save your project

Upgrade your project

Personalise and add more to your project. Perhaps you could change the difficulty level or add more circles to your target.

You could:

- Add a `fourth` and `fifth` circle, in new colours, which score different amounts of points based on their position
- Put emoji in your print messages (**here's a list of emoji** (https://unicode.org/emoji/charts/full-emoji-list.html) you can copy from)
- Make the game easier or harder by changing the `frame_rate(2)` value
- Use `input()` to ask the user which difficulty level they want to play at



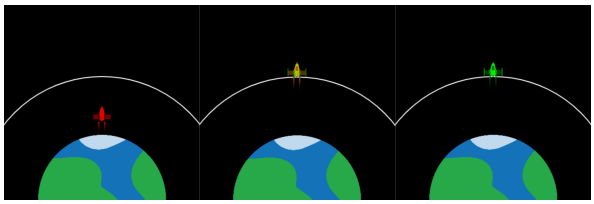
Completed project

You can view the **completed project here** (https://trinket.io/python/f686c82d8a).

Save your project

What next?

If you are following the **Introduction to Python** (https://projects.raspberrypi.org/en/raspberrypi/python-intro) pathway, you can move on to the **Rocket launch** (https://projects.raspberrypi.org/en/projects/rocket-launch) project. In this project, you will make an interactive animation of a rocket launching a satellite into orbit.



If you want to have more fun exploring Python, then you could try out any of **these projects** (https://projects.raspberrypi.org/en/projects?software%5B%5D=python).
