Mapping data

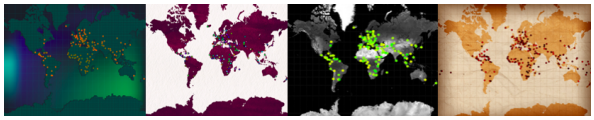
Use Python to make an interactive map that lets users learn interesting facts about the world

Step 1 You will make

Use Python to make an interactive map that lets users learn interesting facts about the world.

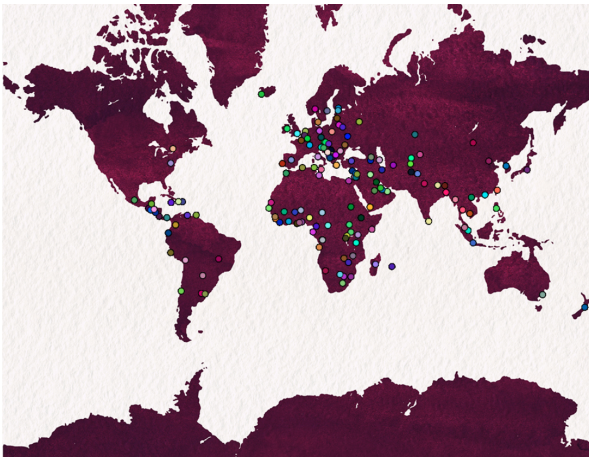
You will:

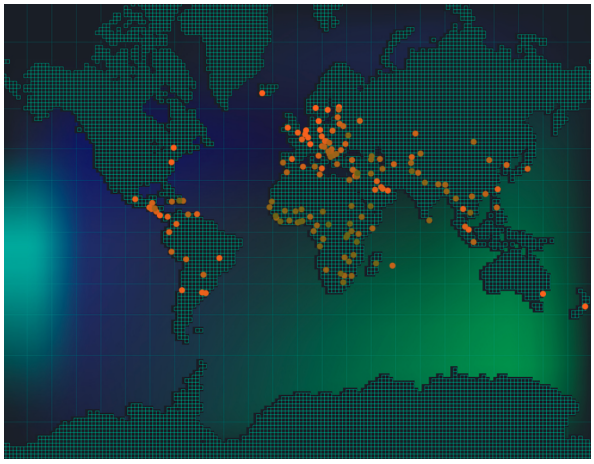
- Use lists and dictionaries to store data
- Use functions and parameters to keep your code clean
- Use code to quickly explore large amounts of data

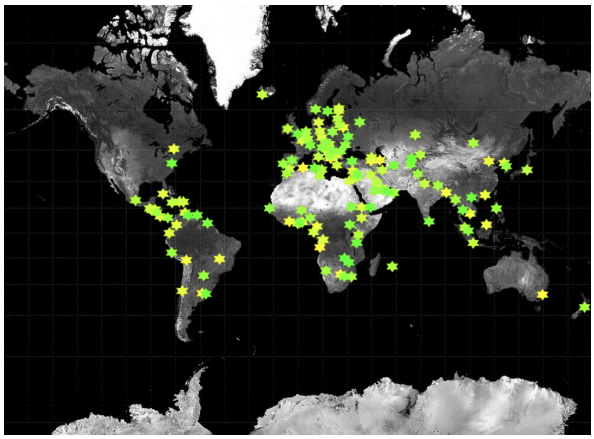


Get ideas

You are going to make some design decisions about what data you want to show to your users, as well as what style of map and pins you will use to display that data.

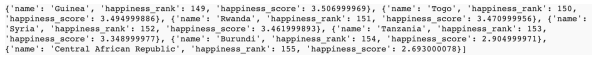






Step 2 Choose and load a data set

Do you have an idea of the kind of display you want to create? Use this step to choose your data and load it into dictionaries. Later, you'll use those dictionaries to build your map.



Open the starter project (https://trinket.io/library/trinkets/c88800b7a4). Trinket will open in another browser tab.

Before you can put your data on a map, you'll need to choose some data to display.

Choose: There are a few CSV files included in the starter project. Read their descriptions below. Then note the name of the file you'd like to use in your display.

> CSV files are Comma-Separated Values files. They contain data in rows and columns, like a table. Each line is a row, with commas separating that row's values into columns.
>
> ```
> United States,2399
> Russia,1413
> Great Britain,1304
> France,780
> Germany,671
> ```

Olympic host nations

File name: `olympics.csv`

This data is a list of regions that have hosted the Olympic Summer Games. The columns of the data are:

- The name of the region
- The number of times they have hosted the games

Here is an example of the data in this file:

```
United States,4
United Kingdom,3
Greece,3
```

World population

File name: `pop.csv`

This data is about populations around the world. The columns of the data are:

- The name of the region
- How many people live in the region
- How many people live in each square kilometre of the region
- The average age of people in the region
- The percentage of people in that region who live in cities

Here is an example of the data in this file:

```
Afghanistan,38928346,60,18,25
Albania,2877797,105,36,63
Algeria,43851044,18,29,73
```

Carbon emissions

File name: `carbon.csv`

This data is about carbon emissions around the world. The numbers in the file are for a single year. The columns of the data are:

- The name of the region
- How much carbon each region emits in total (in thousands of tons)
- How much carbon each region emits, per person that lives in that region (in tons)

Here is an example of the data in this file:

```
Albania,4342.011,1.511
Algeria,130493.653,3.158
Angola,18021.394,0.605
```

Threatened species

File name: `species.csv`

This data is about the number of plant and animal species that are under threat in each region. The columns of the data are:

- The name of the region
- The number of threatened species in that region

Here is an example of the data in this file:

```
Afghanistan,42
Albania,130
Algeria,135
```

National wealth

File name: `gdp.csv`

The gross domestic product (GDP) of a region measures the size of its economy. Regions with larger GDPs are usually richer. The columns of the data are:

- The name of the region
- The total GDP of that region

Here is an example of the data in this file:

```
Aruba,3056424581
Afghanistan,18869945678
Angola,1.22124E+11
```

World happiness

File name: `happy.csv`

This data is from the world happiness report. The report is a survey of the happiness of people in different regions. People were asked to score their happiness on a scale of 0–10. The columns of the data are:

- The name of the region
- Where that region ranks in the world for average happiness
- The average happiness score for the region

Here is an example of the data in this file:

```
Norway,1,7.537000179
Denmark,2,7.521999836
Iceland,3,7.504000187
```

Now that you have picked your data, you need to load it into your program.

Define a `load_data()` function to take a `file_name` variable. Have your function open that file and `print()` out every line in it.

Parameters in functions

When you call or define a function, you always add curved brackets after its name. Just like this example below:

```
def menu(): # Defines a function
  print('Hello')

menu() # Calls a function
```

Those brackets can be used to pass data into a function from another section of your code. This data can then be used by the function to carry out some tasks.

The labels inside the brackets are called parameters. A function can have multiple parameters depending on the purpose of the function.

The example function below has two parameters, which are `name` and `player_id`.

```
def menu(name, player_id):
  print(f'Hello {name}, your player ID is {player_id}')
```

Another part of your code might ask for a player's name or generate a player ID. These can then be passed into the `menu()` function to be used to display a welcome message.

Values that are passed into a function are called arguments.

In the example code below you can see the `menu()` function being defined. You can also see the player's name and ID being passed into the function as arguments.

```
1  def menu(name, player_id):
2
3    print(f'Hello {name}, your player ID is {player_id}') # The function uses the values
4
5  username = 'Hayden'
6  id = 3215
7
8  menu(username, id) # 'Hayden' and '3215' are passed as arguments into the function
```

main.py — load_data()

```
    # Put code to run when the mouse is pressed here
    def mouse_pressed():
      pixel_colour = color(get(mouse_x, mouse_y))

    def load_data(file_name):
      with open(file_name) as f:
5       for line in f:
          print(line)
```

Tip: You will be moving data around a lot in the next few steps. It's a good idea to `print()` everything out. This will help you understand what your data looks like at each step. It's also good for catching bugs. You can comment the `print()` lines out later (with `#`).

Add a call to `load_data()` in your `setup()` function. Pass it the name of the data file you chose above. You can check the list below if you need a reminder of the file name.

- Olympic host nations — `olympics.csv`
- World population — `pop.csv`
- Carbon emissions — `carbon.csv`
- Threatened species — `species.csv`
- National wealth — `gdp.csv`
- World happiness — `happy.csv`

Test: Run your program. Check the data that prints out in the output area.

Debug: You might get an error message about your file name being 'not defined'. If you do, check that you have put the name in quotes when you call the `load_data()` function. For example, `load_data('pop.csv')`.

Now the data is loaded, you need to get the data for each region and break it into a list. Then you can load that list into a dictionary.

Add code to your `load_data()` function to use the `split()` function to break each line into a list. Call that list `info`.

Split a text string into a list

The `split()` function breaks a string into a list. `split(',')` makes a new list item every time it sees a comma. So,

```
info = 'Estonia,1326535,31,42,68'
my_list = info.split(',')
```

would put `['Estonia', '1326535', '31', '42', '68']` into `my_list`.

main.py — load_data()

```
def load_data(file_name):
  with open(file_name) as f:
    for line in f:
      #print(line)
      info = line.split(',')
```

Now use the list you made from each region's data to create a dictionary for each region. Include the name of the region and the numbers you want to use in your display.

Add code to your `load_data()` function that converts the data you've chosen into a dictionary.

Use `print()` to check the dictionaries look like you expect.

Field names for the csv files

Olympic host nations — `olympics.csv`

- The name of the region
- The number of times they have hosted the games

World population — `pop.csv`

- The name of the region
- How many people live in the region
- How many people live in each square kilometre of the region
- The average age of people in the region
- The percentage of people in that region who live in cities

Carbon emissions — `carbon.csv`

- The name of the region
- How much carbon each region emits in total (in thousands of tons)
- How much carbon each region emits, per person that lives in that region (in tons)

Threatened species — `species.csv`

- The name of the region
- The number of threatened species in that region

National wealth — `gdp.csv`

- The name of the region
- The total GDP of that region

World happiness — `happy.csv`

- The name of the region
- Happiness rank
- Happiness score

main.py — load_data()

```
def load_data(file_name):
  with open(file_name) as f:
    for line in f:
      #print(line)
      info = line.split(',')
      # Change the dictionary to match the data you're using
      region_dict = {
        'name': info[0],
        'happiness rank': info[1],
        'happiness score': info[2]
      }
      print(region_dict)
```

Test: Run your code and check that the dictionaries it prints out look like you expect them to: a 'name' key with a text string for a value, and whatever keys and values you expect based on your code.

Debug: If you see a message about `list index out of range`, check that you are trying to load the right number of values into your region dictionary. This may be a different number of values to the example code above. You should also use key names that match the data you chose.

Now your `load_data()` function creates dictionaries for each region. You need to store those dictionaries somewhere the rest of your program can get them. A list is a good choice.

Create an empty list called `region_list`.

main.py

```
#!/bin/python3
from p5 import *
from regions import get_region_coords

region_list = []
```

In `load_data()`, add each of your dictionaries to `region_list` using `append`. This will let you work with the data in the rest of your program.

main.py - load_data()

```
def load_data(file_name):
  with open(file_name) as f:
    for line in f:
      info = line.split(',')
      region_dict = {
        'name': info[0],
        'happiness rank': info[1],
        'happiness score': info[2]
      }
      #print(region_dict)
      region_list.append(region_dict)
```

Add a line in your `setup()` function that prints the `region_list` out.

Test: Run your program and check that it prints out a list of dictionaries. It should look something like this:

```
{'name': 'Guinea', 'happiness_rank': 149, 'happiness_score': 3.506999969}, {'name': 'Togo', 'happiness_rank': 150,
'happiness_score': 3.494999886}, {'name': 'Rwanda', 'happiness_rank': 151, 'happiness_score': 3.470999956}, {'name':
'Syria', 'happiness_rank': 152, 'happiness_score': 3.461999893}, {'name': 'Tanzania', 'happiness_rank': 153,
'happiness_score': 3.348999977}, {'name': 'Burundi', 'happiness_rank': 154, 'happiness_score': 2.904999971},
{'name': 'Central African Republic', 'happiness_rank': 155, 'happiness_score': 2.693000078}]
```

Tip: Like the other `print()` statements you've used, you can comment this line out once you've used it for testing and your code works as expected.

Debug: You might find some bugs in your project that you need to fix. Here are some common bugs.

My code doesn't run

Check your code is properly indented. The code under the `with` should be indented, and then code under the `for` loop should be indented again.

I get a message that the csv file is 'not defined'

Check your call to `load_data()` to be sure that the name of the file is a string.

My info list just has one big item in it

Check that you have `','` in the `()` of `line.split()`

I get a message that split is 'not defined'

If you see a message about `split` being 'not defined', check that you have included `line.` before it.

I get a message that region_list is 'not defined'

If you see a message about `region_list` being 'not defined', check that you have created it as an empty list — with `region_list = []` — before trying to add things to it

Save your project

Step 3 Pick a map and pins

Choose how you'll display the data you've selected.

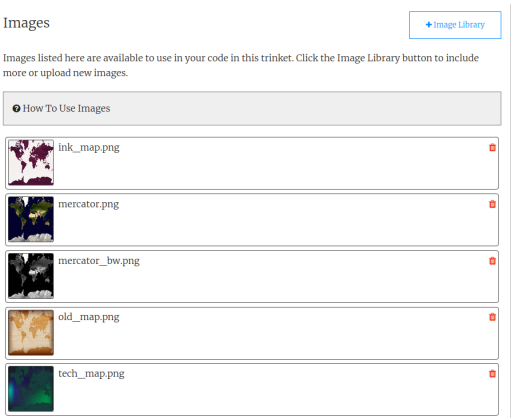
Add code to your `setup()` function to set the size of your canvas to 991 pixels wide and 768 pixels high.

main.py - setup()

```
# Put code to run once here
def setup():
  load_data('happy.csv')
  size(991, 768)
```

Think about how you want to display the data you've picked: what kind of map do you want to use?

- ink-map.png
- mercator.png
- mercator_bw.png
- old-map.png
- tech-map.png



Choose: The starter project includes several map images. Pick one you like, and load the image in your `setup` function.

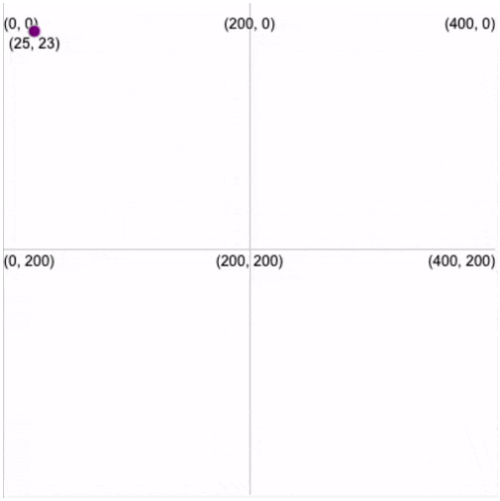
main.py - setup()

```
# Put code to run once here
def setup():
  load_data('happy.csv')
  size(991, 768)
  map = load_image('mercator.png') # Replace with your image
```

Add code to your `setup()` function to draw the map so it covers the whole canvas.

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.



main.py - setup()

```
def setup():
# Put code to run once here
  load_data('happy.csv')
  size(991, 768)
  map = load_image('map.png') # Replace with your image
  image(
      map, # The image to draw
      0, # The x of the top-left corner
      0, # The y of the top-left corner
      width, # The width of the image
      height # The height of the image
      )
```

Test: Run your program and look at your map! You will probably need to switch to the fullscreen view to see the whole map.

Choose: What shape of pin will you place in each location? Your pin will need to be a single colour so that it is easy for a user to click on.

You could choose a single shape, such as:

- A circle
- A square
- A triangle

Or you could create a pin out of multiple geometric shapes, such as:

- A heart
- A map pin
- A star

Define a function called `draw_pin`. It should draw a pin, of your own design, on the map. It should take three parameters:

- The x coordinate for the pin.
- The y coordinate for the pin.
- The colour of the pin. This should be a `p5` `color()`.

main.py - draw_pin()

```
def draw_pin(x, y, colour):
  # Put code to draw your pin here
```

As you create your `draw_pin` function, call it to see how it appears on the screen. You should call your `draw_pin` function from the `setup()` function.

You can use the arguments shown below to place a `red` pin the middle of the screen.

main.py - setup()

```
def setup():
# Put code to run once here
  size(991, 768)
  map = load_image('map.png') # Replace with your image
  image(
    map, # The image to draw
    0, # The x of the top-left corner
    0, # The y of the top-left corner
    width, # The width of the image
    height # The height of the image
    )
  draw_pin(300, 300, color(255,0,0))
```

Parameters in functions

When you call or define a function, you always add curved brackets after its name. Just like this example below:

```
def menu(): # Defines a function
  print('Hello')

menu() # Calls a function
```

Those brackets can be used to pass data into a function from another section of your code. This data can then be used by the function to carry out some tasks.

The labels inside the brackets are called parameters. A function can have multiple parameters depending on the purpose of the function.

The example function below has two parameters, which are `name` and `player_id`.

```
def menu(name, player_id):
  print(f'Hello {name}, your player ID is {player_id}')
```

Another part of your code might ask for a player's name or generate a player ID. These can then be passed into the `menu()` function to be used to display a welcome message.

Values that are passed into a function are called arguments.

In the example code below you can see the `menu()` function being defined. You can also see the player's name and ID being passed into the function as arguments.

```
def menu(name, player_id):

  print(f'Hello {name}, your player ID is {player_id}') # The function uses the values

username = 'Hayden'
id = 3215

menu(username, id) # 'Hayden' and '3215' are passed as arguments into the function
```

Colours in p5

The `p5` `color()` function expects three numbers: one each for red, green, and blue.

```
blue = color(92, 204, 206) #Red = 92, Green = 204, Blue = 206
```

You can use the `fill()` function to fill a shape with colour. `fill()` applies to every shape drawn after it.

```
green = color(149, 212, 122)
fill(green)
rect(0, 250, 400, 150) # This shape will be filled with the colour
```

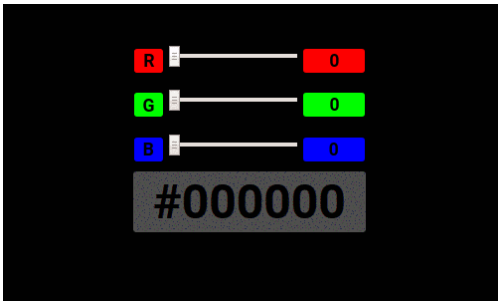
To remove fills completely, call `no_fill()` before drawing your shape(s).

You can set a colour for the border around a shape with the `stroke()` function:

```
white = color(255, 255, 255)
stroke(white)
rect(0, 250, 400, 150) # This shape will have a white border
```

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

Red	Green	Blue	Colour
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).

Draw an ellipse

Draw a circle or ellipse using: `ellipse(x, y, width, height)`

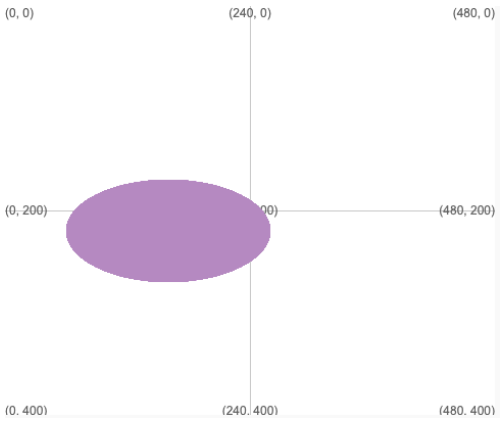
The ellipse will be drawn using the stroke and fill values that have been set before `ellipse` is called.

main.py

```
ellipse(160, 220, 200, 100) # x, y, width, height
```

The ellipse will be centered at the (x, y) coordinates given by the first two numbers.

The third number is the width and the fourth is the height of the ellipse.



Make the width and height the same to draw a circle.

Draw a rectangle

Draw a square or rectangle using: `rect(x, y, width, height)`

The rectangle will be drawn using the stroke and fill values that have been set before `rect` is called.

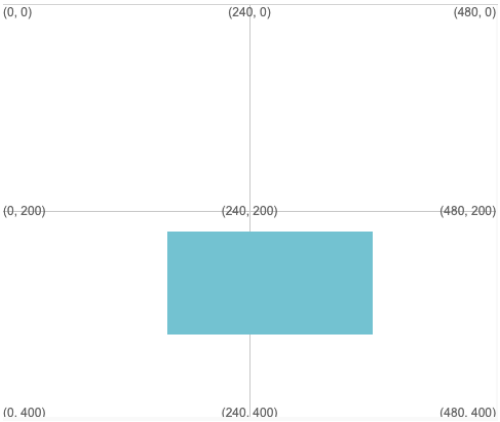
main.py

```
rect(160, 220, 200, 100) # x, y, width, height
```

The rectangle will be drawn with its top left corner at the (x, y) coordinates given by the first two numbers.

Tip: If you want the center of the rectangle to be at the (x, y) coordinates then call `rect_mode(CENTER)` in the `setup` function.

The third number is the width and the fourth is the height of the rectangle.



Make the width and height the same to draw a square.

Draw a triangle

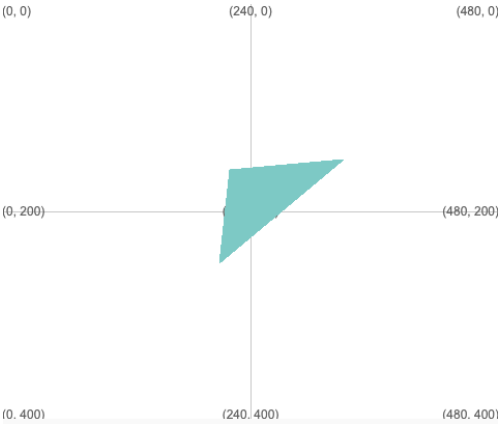
Draw a triangle using: `triangle(x1, y1, x2, y2, x3, y3)`

The triangle will be drawn using the stroke and fill values that have been set before `triangle` is called.

main.py

```
triangle(210, 250, 330, 150, 220, 160) # (x1, y1), (x2, y2), (x3, y3)
```

The triangle will be drawn with a corner at each of the three coordinates given by (x1, y1), (x2, y2), (x3, y3).



Tip: Your `draw_pin` function can make other shapes out of these basic ones.

Debug: You might find some bugs in your project that you need to fix. Here are some common bugs.

My map isn't loading

Check the filename really carefully — remember capital letters are different to lower-case letters and punctuation is important.

My map is the wrong size

Check the inputs that control the width and height of the image:

```
image(
    map, # The image to draw
    0, # The x of the top-left corner
    0, # The y of the top-left corner
    width, # The width of the image
    height # The height of the image
    )
```

My pin isn't appearing

Make sure that you have called the `draw_pin()` function in your `draw()` function, and passed it the values it needs. For example:

main.py - draw()

```
draw_pin(width/2, height/2, color(255,0,0))
```

Also, make sure you are calling `draw_pin()` after you call `image()` to create the background. If not, you're drawing the map over the pin!

Save your project

Step 4 Mark your data

Display your data on the map, and make it interactive.

Before you can put pins on the map for each place you have data about, you need to know where those places are. The starter project includes code to give you those locations.

You can use `get_region_coords()` to return a dictionary of the coordinates for a region. For example `get_region_coords('Japan')` will return `{'x': 880.151122422, 'y': 278.639809465}`.

Define a `draw_data()` function to put your data on the map. At first you can just print out the region's name and its `x` and `y` coordinates.

It should loop through your `region_list` and print a line for each region.

main.py — draw_data()

```
def draw_data():
  for region in region_list:
    region_name = region['name'] # Get the name of the region
    region_coords = get_region_coords(region_name) # Use the name to get coordinates
    region_x = region_coords['x'] # Get the x coordinate
    region_y = region_coords['y'] # Get the y coordinate
    print(region_name, region_x, region_y)
```

In your `setup()` function, comment out your `draw_pin()` code and instead call `draw_data()`.

main.py - setup()

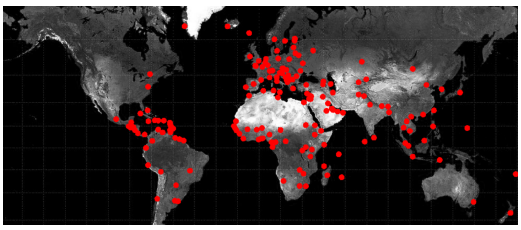
```
  def setup():
  # Put code to run once here
    size(991, 768)
    map = load_image('map.png') # Replace with your image
    image(
      map, # The image to draw
      0, # The x of the top-left corner
      0, # The y of the top-left corner
      width, # The width of the image
      height # The height of the image
      )
    # draw_pin(300, 300, color(255,0,0))
    draw_data()
```

Instead of printing out the name of the region, and its coordinates, you can use your `draw_pin()` function to place your pins on the map. The code below colours the pins red (`color(255, 0, 9)`), but you can choose a different colour.

main.py — draw_data()

```
def draw_data():
  for region in region_list:
    region_name = region['name'] # Get the name of the region
    region_coords = get_region_coords(region_name) # Use the name to get coordinates
    region_x = region_coords['x'] # Get the x coordinate
    region_y = region_coords['y'] # Get the y coordinate
    #print(region_name, region_x, region_y)
    region_colour = color(255, 0, 0) # Set the pin colour
    draw_pin(region_x, region_y, region_colour) # Draw the pin
```

Test: Run your program. You should see lots of pins pop up on your map! Depending on the data you chose, you might see more or fewer pins than in the image below.



Next, you need to add some code to let users click on a pin and see some information printed out. To do this, each pin needs to be a different colour, and you need a way to match those colours to the right data.

Choose: Every pin needs a unique colour. But there are lots of different ways to make this happen. Here are a few suggestions, but you can create your own.

Change the value of one colour

This example changes the value for red each time the code places a pin:

main.py — draw_data()

```
def draw_data():
  red_value = 255 # Set a starting value for red

  for region in region_list:
    region_name = region['name']
    region_coords = get_region_coords(region_name)
    region_x = region_coords['x']
    region_y = region_coords['y']
    region_colour = color(red_value, 0, 0) # Use the red value in the colour
    draw_pin(region_x, region_y, region_colour)
    red_value -= 1 # Change the red value
```

Change the value of multiple colours

This example changes the red, green, and blue values each time the code places a pin:

main.py — draw_data()

```
def draw_data():
  red_value = 255 # Set a starting value for red
  blue_value = 0
  green_value = 255
  for region in region_list:
    region_name = region['name']
    region_coords = get_region_coords(region_name)
    region_x = region_coords['x']
    region_y = region_coords['y']
    region_colour = color(red_value, green_value, blue_value) # Use all the colours
    draw_pin(region_x, region_y, region_colour)
    red_value -= 1 # Change the red value
    green_value += 1 # Change the green value
    blue_value -= 1 # Change the blue value
```

Choose random colours

At the top of your code, with your other imports, you will need to import `randint` from the `random` library.

You can then choose a random colour for your region colours; a different colour will be picked each time the `for` loop is executed. There is a small chance that two or more colours might end up the same, but it is a very small chance.

main.py — draw_data()

```
from random import randint

def draw_data():
  for region in region_list:
    region_name = region['name']
    region_coords = get_region_coords(region_name)
    region_x = region_coords['x']
    region_y = region_coords['y']
    region_colour = color(randint(0,255), randint(0,255), randint(0,255)) # Select a random colour
    draw_pin(region_x, region_y, region_colour)
```

Test: Run your program and check that the pins are different colours. If you don't have many pins, it may be hard to tell. In that case, try using bigger changes between each pin.

Your map has unique pins for each location, but you need to add some code to connect those pins to the information you want to show your users.

To use the pin's colour to look up the information, you need to create a dictionary to store the colours and link them to the region.

main.py

```
#!/bin/python3
from p5 import *
from regions import get_region_coords
from random import randint

region_list = []
colours = {}
```

As the pins are placed, the `region` can be stored in the dictionary along with the colour of the pin.

main.py

```
def draw_data():
  red_value = 255
  for region in region_list:
    region_name = region['name'] # Get the name of the region
    region_coords = get_region_coords(region_name) # Use the name to get coordinates
    region_x = region_coords['x'] # Get the x coordinate
    region_y = region_coords['y'] # Get the y coordinate
    region_colour = color(i, 100, 0) # Set the pin colour
    colours[region_colour] = region
    draw_pin(region_x, region_y, region_colour)
    red_value -= 1
```

When the user clicks on a pin, the colour of the pin is retrieved, and then the corresponding region is found in the dictionary.

In your `mouse_pressed()` function, lookup the `pixel_colour` in the `colours` dictionary and print out the `region`.

Remember that `colours` is a dictionary of dictionaries. You will have to get the dictionary of region information, then get the information from inside that dictionary. For example:

main.py

```
def mouse_pressed():
# Put code to run when the mouse is pressed here
  pixel_colour = color(get(mouse_x, mouse_y))
  facts = colours[pixel_colour]
  print(facts['region'])

```

It's important to check if a key is in a dictionary. If you click on an area of the map without a pin, you will receive a `KeyError`.

You can check if a value is in a dictionary by using `in`:

main.py

```
def mouse_pressed():
# Put code to run when the mouse is pressed here
  pixel_colour = color(get(mouse_x, mouse_y))
  if pixel_colour in colours:
    facts = colours[pixel_colour]
    print(facts['region'])
  else:
    print('Region not detected')

```

Test: Run your program. Click on a pin and check that your program correctly prints out data about that area.

You can print out other facts about the region you clicked on by adding more `print()` statements. This will depend on the data set that you used. If you used `happy.csv` for instance, you could print the following:

main.py

```
def mouse_pressed():
# Put code to run when the mouse is pressed here
  pixel_colour = color(get(mouse_x, mouse_y))
  if pixel_colour in colours:
    facts = colours[pixel_colour]
    print(facts['region'])
    print(facts['happiness_rank']) # Your first data fact
    print(facts['happiness_score']) # Your second data fact
  else:
    print('Region not detected')

```

Debug: You might find some bugs in your project that you need to fix. Here are some common bugs.

My pins do not appear on the map

If your pins are not appearing on the map, check that you are calling your `draw_data()` function from your `draw()` function.

I get a message about a 'KeyError'

If you get a message about 'KeyError', check that the spelling of your dictionary keys match when you put values in and when you read them out. Whether the letters are UPPER CASE or lower case is important too.

If the error is for the `colours` dictionary, make sure you check the key exists in `colours` before trying to get the value.

It keeps displaying 'Region not detected'

Your mouse click needs to be in the centre of your pin to make sure that it detects the correct colour.

Try clicking closer to the centre of your pin.

Save your project

Upgrade your project

If you have time you can upgrade your project.

Here are some ideas you could try:

- Use pins to display data — change the size or shape of the pin based on some value in the region's data. You can combine changes in shape and size to show even more.
- Filter the data — use `if` statements to only show pins on the map that meet conditions you choose. For example, only showing places where less than half the people live in cities. For an extra upgrade, let the user choose which filters to use.
- Add a second data file — load another file of data and display it using different pins. Can you use some of your existing functions if you make some small changes? Do you want to use two sets of pins, or put all the data together on one pin?

Multicoloured pins: Population size and carbon emissions (see inside (https://trinket.io/python/b51eb6b362))

This projects uses small multicoloured pins and uses both the population and carbon emissions data sets.

You should use the link above to view this project. Expand it into full screen mode to be able to see and correctly click on pins.

Data-sized pins: Olympic host countries (see inside (https://trinket.io/python/42df9879d7))

This project has been upgraded to show larger pins for areas that have hosted the Olympic Games more often.

You should view this project in full screen mode to see the whole map. You may also need to resize the map window.

Filter data: Global population — rural and urban (see inside (https://trinket.io/python/afcdf3fdea))

This project has been upgraded to allow you to filter the data used to make the display.

You should view this project in full screen mode to see the whole map. You may also need to resize the map window.

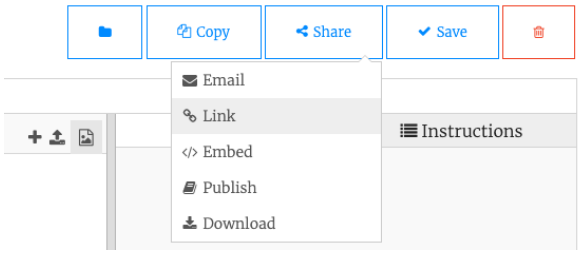
Save your project

Share

If you are in a club, why not share your project with friends?

You could also show your family how your project works by sending them a link.

To get a link, go to the Share menu button in the upper-right corner of your Trinket's edit page and select Link.



Tip: You can share a link to your project even if you don't have a Trinket account; however, without an account, the link to your project will change each time you update it. If you have shared the link with someone, you will need to send a new link for them to see the changes.

Inspire the Raspberry Pi Foundation community with your project!

To submit your project to our 'Mapping data - Community' (https://wke.lt/w/s/E9LnpL) studio, please complete this form (https://form.raspberrypi.org/f/community-project-submissions).

What next?

If you are following the More Python (https://projects.raspberrypi.org/en/raspberrypi/more-python) path, you can move on to the Persuasive presentation (https://projects.raspberrypi.org/en/projects/persuasive-data-presentation) project. In this project, you will make a visual presentation of your own design, based on data you choose.

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).
